

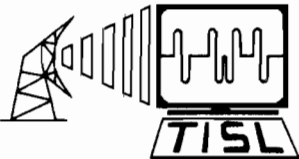
Integrating LAN Data and Digital Voice
on Wire Twisted Pair

Timothy R. Davis
Victor S. Frost

Telecommunications and Information Sciences Laboratory
University of Kansas
Lawrence, Kansas 66045

Technical Report 6731-6
February 1987

Supported by
National Science Foundation
Presidential Young Investigator Award
and
AT&T Information Systems



TELECOMMUNICATIONS AND INFORMATION SCIENCES LABORATORY
THE UNIVERSITY OF KANSAS CENTER FOR RESEARCH, INC.
2291 Irving Hill Drive - Campus West Lawrence, Kansas 66045

ABSTRACT

A new approach for integrating PCM voice and LAN data on twisted pair is described. The motivation for developing this technique came from a need to reduce the number of twisted pairs required to communicate with a voice/data terminal located on a user's desk. Typically, five or more pairs are required: two for access to LAN; two for digital voice; one for power and ground. Traditional solutions employ TDM-PCM, FDM or Packet Switching. These methods suffer from synchronization, bandwidth limitation and complexity problems respectively. The number of pairs can be reduced using a waveform level integration of the high rate data and digital voice (or DCP) without suffering from the above problems.

The integration technique utilizes a new baseband coding system. Manchester encoded data sources, one at 64/128/160 kbps and one at 1024 kbps, drive a multiplexer which produces a three level waveform at the receiver. The receiver generates the three level signal by taking the difference of two received voltages thus providing excellent noise immunity. The multiplexer output can be transformer coupled to the receiver allowing power to be transmitted with the signal. The self-clocking qualities of the original manchester encoded signals are also maintained for increased clock recovery performance.

Recovery of the sources from the three level waveform requires only a schmitt trigger and an absolute value circuit, making the hardware quite simple.

The hardware implementation has been completed and successfully demonstrated over 250 feet of twisted pair at rates of 64/128+1024 kbps and over 100 feet for 160+1024 kbps operation. Operation at 750-800ft for 64/128+1024 is expected soon. In addition, the work done indicates the possibility of multiplexing two 1024 KBPS sources on one twisted pair.

CONTENTS

Abstract	i
Acknowledgments	ii
Contents	iii
List of Figures	v
List of Tables	vii
1. INTRODUCTION.....	1
1.1 A Question.....	1
1.2 Background.....	1
1.3 Problem Description.....	2
1.4 Motivation.....	2
1.5 Research goal.....	4
1.6 Traditional solutions.....	4
1.7 The rest of the story.....	13
2. A NEW BASEBAND MULTIPLEXER APPROACH.....	15
2.1 Analog Sum Multiplexer.....	15
2.2 Analog Product.....	18
2.3 DVM Signal Space Representation.....	21
2.4 Baseband Implementation.....	24
3. Data Voice Multiplexer Design.....	26
3.1 Overview.....	26
3.2 Voice and Data sources.....	26
3.3 Multiplexer Product Implementation.....	29
3.4 The Channel.....	32
3.5 The DVM demultiplexer.....	32
3.6 Manchester decoders.....	35
3.7 DVM signal properties.....	37
4. SIMULATING THE DATA AND VOICE MULTIPLEXER SYSTEM.....	39
4.1 Simulation Purpose.....	39
4.2 Simulation Organization.....	40
4.3 Simulation Design.....	42
4.4 Simulation Results.....	45
4.5 Simulation Conclusions.....	61
5. Data and Voice Multiplexer Prototype Hardware.....	62
5.1 POP - Principles of Operation.....	62
5.2 Performance.....	70
6. Operation and Performance at DCP Rate.....	73
6.1 The Vee Problem.....	73
6.2 Conclusion About DCP.....	75

7. Conclusion.....	77
7.1 Summary of Results.....	77
8. REFERENCES.....	80
 Appendix	
A. SYSTID SIMULATION CODE.....	81
A.1 DAVIS06.....	81
A.2 DAVIS07.....	87
A.3 TWISTED PAIR CHANNEL.....	93
A.4 D FLIP FLOP.....	98
A.5 4-1 MULTIPLEXER.....	99
A.6 SCHMITT TRIGGER.....	100
A.7 TIMER.....	102
A.8 RANDOM PULSE GENERATOR.....	103
A.9 ABSOLUTE VALUE.....	104
A.10 EXCLUSIVE OR.....	105
A.11 TRIMPOWER.FOR.....	106
A.12 BANDPASS.FOR.....	108
A.13 TXLINE.....	109
A.14 RUN06.COM.....	111
A.15 RUN07.COM.....	111
 B PROTOTYPE CIRCUIT SCHEMATICS.....	 112
C. PARTS LOCATER.....	121
D. SIGNAL LOCATER.....	127

LIST OF FIGURES

Figure 1-1.	Data and voice multiplexer solution.....	3
Figure 1-2.	Time Division Multiplexing (TDM).....	6
Figure 1-3.	Example of TDM system complexity.....	7
Figure 1-4.	Frame alignment search mechanism.....	10
Figure 1-5.	Voice Over Data (VOD) multiplexer and demultiplexer.....	12
Figure 1-6.	Spectral representation of VOD system.....	12
Figure 2-1.	Analog sum multiplexer and demultiplexer.....	16
Figure 2-2.	Analog sum signal and it's spectrum.....	16
Figure 2-3.	Analog product signal.....	17
Figure 2-4.	Schmitt trigger.....	20
Figure 2-5.	Signal space and signal components for DVM and TDM.....	23
Figure 2-6.	QPSK like implementation of TDM.....	25
Figure 3-1.	General DVM system organization.....	27
Figure 3-2.	Manchester encoded, random binary sources.....	28
Figure 3-3.	DVM multiplexer implementation.....	31
Figure 3-4.	Alternate DVM multiplexer implementation.....	33
Figure 3-5.	DVM Demultiplexer.....	34
Figure 3-6.	Manchester decoders.....	36
Figure 4-1.	Simulation system block diagram.....	41
Figure 4-2.	Twisted pair modeling process.....	43
Figure 4-3.	Prominent features in the eye diagrams.....	48
Figure 4-4.	0.5 MHz eye diagram.....	49

Figure 4-5.	1.0 MHz eye diagram.....	50
Figure 4-6.	1.5 MHz eye diagram.....	51
Figure 4-7.	3.0 MHz eye diagram.....	52
Figure 4-8.	Eye diagram of 64 KBPS signal.....	54
Figure 4-9.	Eye diagram: 22 gauge, 250 ft cable.....	56
Figure 4-10.	Eye diagram: 24 gauge, 250 ft cable.....	57
Figure 4-11.	Eye diagram: 22 gauge, 500 ft cable.....	58
Figure 4-12.	Eye diagram: 24 gauge, 500 ft cable.....	59
Figure 5-1.	Hardware performance with 100ft of cable.....	71
Figure 6-1.	DCP rate 'vee' problem.....	74
Figure 6-2.	Illustration of sample point location for DCP.....	76

LIST OF TABLES

TABLE 3-1.	Demonstration of Manchester Encoding.....	29
TABLE 3-2.	DVM signal product formation.....	30
TABLE 4-1.	Errors vs voice to data clock skew.....	60
TABLE 5-1.	U15 multiplexer input coding.....	66

1. INTRODUCTION

1.1 A Question

Can a new waveform level voice/data multiplexing scheme be developed that is equal or superior to the schemes that exist today? TDM and FDM are well developed with many years of experience behind them. Packetized voice and data seem to be the thrust of mainstream research today. Is a new effort justified? The answer is developed in the sections that follow beginning with some background, continuing with a description of current problems involving multiplexing, and ending with a motivation to solve the problem and provide an answer to our question.

1.2 Background

The age of high speed, networked, digital computers has arrived. Computers are available to virtually everybody. Every year adds applications using the information gathering, processing, and distribution capabilities computers possess. Networking is bringing information to personal computers on desks everywhere. Perhaps, the word computer should be replaced by "Information source." The age should be called "The age of high speed networked information sources."

The desire to network information sources is shaping a trend in building design to include a plan to wire new buildings for power and for cable to connect information management systems. Installation of digital pathways for voice and local area network connections are being demanded by the tenant. Those "pathways" are likely to be twisted pair

phone wire. Twisted pair phone wire is used because it is cheap. The networking/information industry is directing its efforts to take advantage of the twisted pair trend, and their efforts will result in more hardware that works with phone wire.

1.3 Problem Description

Soon, if not already, a computer (or terminal) will take its place beside the phone on every desk. In addition, the phone will become a digital instrument. Soon, local area networks (LANs) will migrate into private branch exchanges (PBXs) and the computer will be switchable to virtually any network. A phone will typically generate 64 KBPS PCM digital voice. The computer will transmit/receive at rate of 1024 KBPS or higher.

The computer and the phone typically require about four to five pairs of wire to connect them to the PBX. Two pair each for duplex operation plus one pair for power and ground. If everybody across the country suddenly gets a computer on their desk, copper wire would become a good investment!

The solution shown in figure 1.1 is to reduce the number of pairs required by using a multiplexer.

1.4 Motivation

The primary motivation to apply multiplexing is to reduce the number of twisted pairs required for voice/data integration. Decreasing the complexity of the multiplexer is also desirable. Reducing the number of twisted pairs required for duplex voice/data communication results in

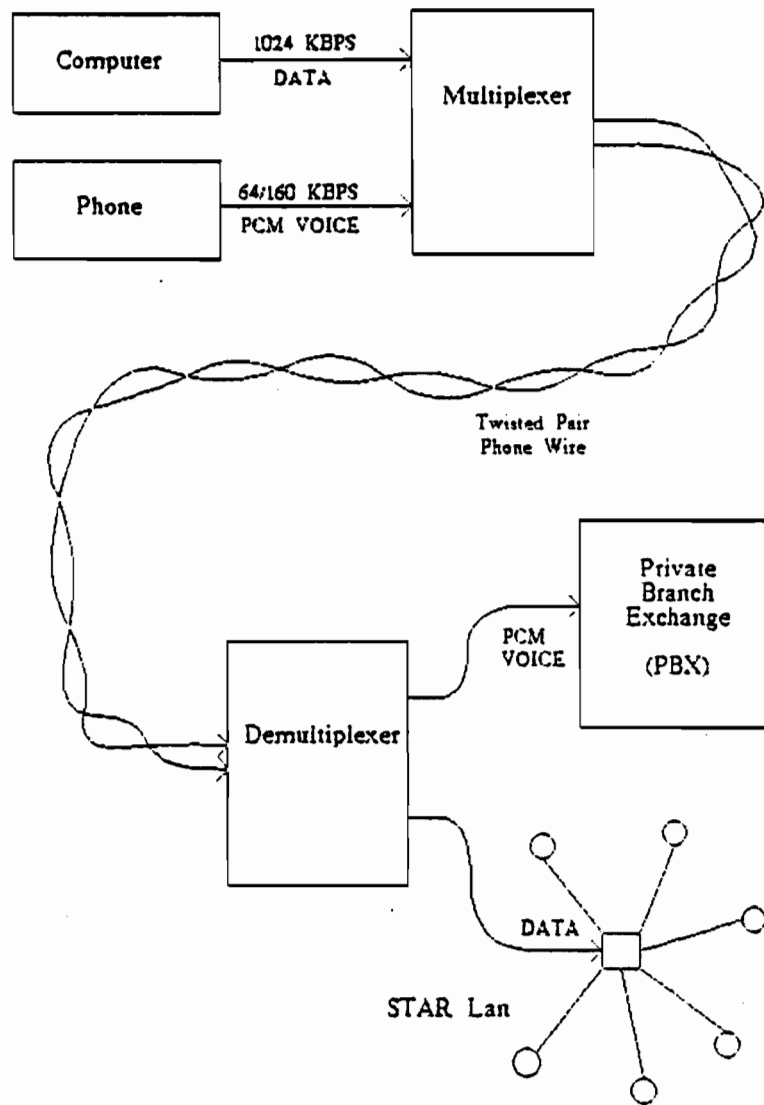


Figure 1-1. Data and voice multiplexer solution.

tremendous saving. Two pairs are required for duplex digital voice, two for a connection to a star local area network (LAN), plus one pair for power and ground; Five pairs in all. It is possible to use only two (maybe three) pairs with a multiplexer. Roughly half the cost to install a new connection to a PBX is the cost of labor [polo86]. A simple, user installable multiplexer would reduce the need to add extra wire for expansion to voice/data workstations, eliminating expensive labor costs.

1.5 Research goal

The goal of this research is to implement a multiplexer that integrates a PCM voice source and a high speed LAN data source at 1024 KBPS on wire twisted pair. The voice channel would carry a single 64 KBPS source or might be capable of carrying AT&T's Digital Communications Protocol (DCP). DCP carries two 64 KBPS voice channels plus signaling for a total rate of 160 KBPS. The data would come from a local area network connection going to a terminal or personal computer (PC) on the user's desk.

1.6 Traditional solutions

Traditional multiplexing solutions rely on time division multiplexing (TDM) or frequency division multiplexing (FDM). These schemes can be classified as waveform level integration techniques. They operate without knowledge of the type of data they carry. Approaches such as packet switching work on top of some existing service and generally use knowledge about the statistical properties of the data being transmitted to determine the transmission protocol. They are implemented with the

help of additional (fairly complex) hardware using existing digital transmission facilities. Since a simple solution is sought so that simple hardware can be implemented, packet switching will not be considered. A description of a packet voice/data multiplexer is given in [baum86]. Waveform level multiplexing is the approach that will be taken for solving our multiplexing problem. Next we will describe some of the problems with traditional the solutions.

1.6.1 Time division multiplexing (TDM). In TDM, every signal occupies a small portion of the time needed to transmit pieces of all the signals. Hence, the name time division multiplexing. A simple, low speed TDM system can be constructed using a motor driven rotating switch that repeatedly samples several signals at regular intervals. To reconstruct the original signals, a similar switch is used in the receiver that is synchronized to the rotating switch in the transmitter. The whole operation works reasonably well when the switches are synchronized.

In practice an electronic method is used to switch one of the inputs to the output as shown in figure 1.2 with a precise timing reference controlling the when the switching occurs.

If the signals are quantized, converted to binary using an analog to digital converter (A/D), then sent in binary form one word at a time using TDM, then we call the system TDM pulse code modulation (TDM/PCM). A TDM/PCM system is shown in figure 1.3. This system is complex. The diagram contains quite a few sophisticated blocks. Ignoring the hardware for generating the PCM voice, look at the additional blocks

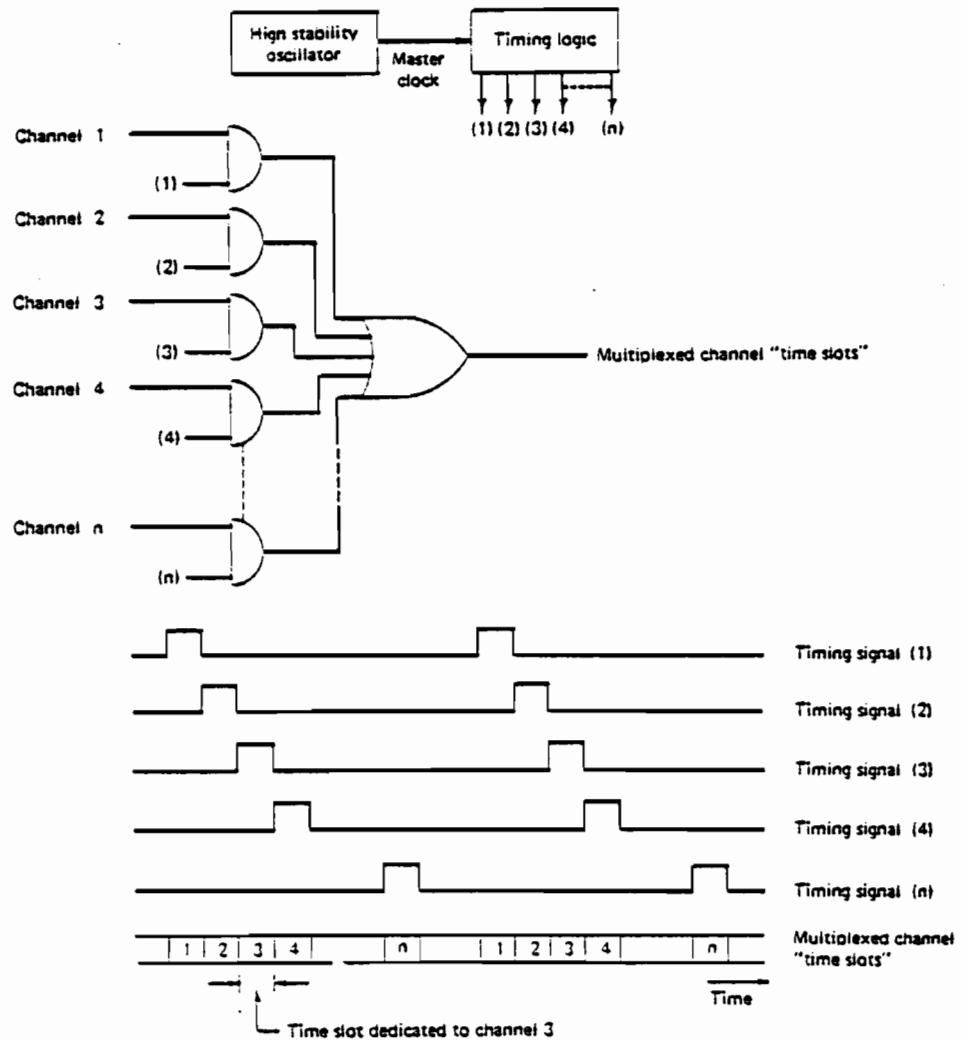


Figure 1-2. Time Division Multiplexing (TDM).
 (From PCM and Digital Transmission Systems by Frank F.E. Owen; Pg118)

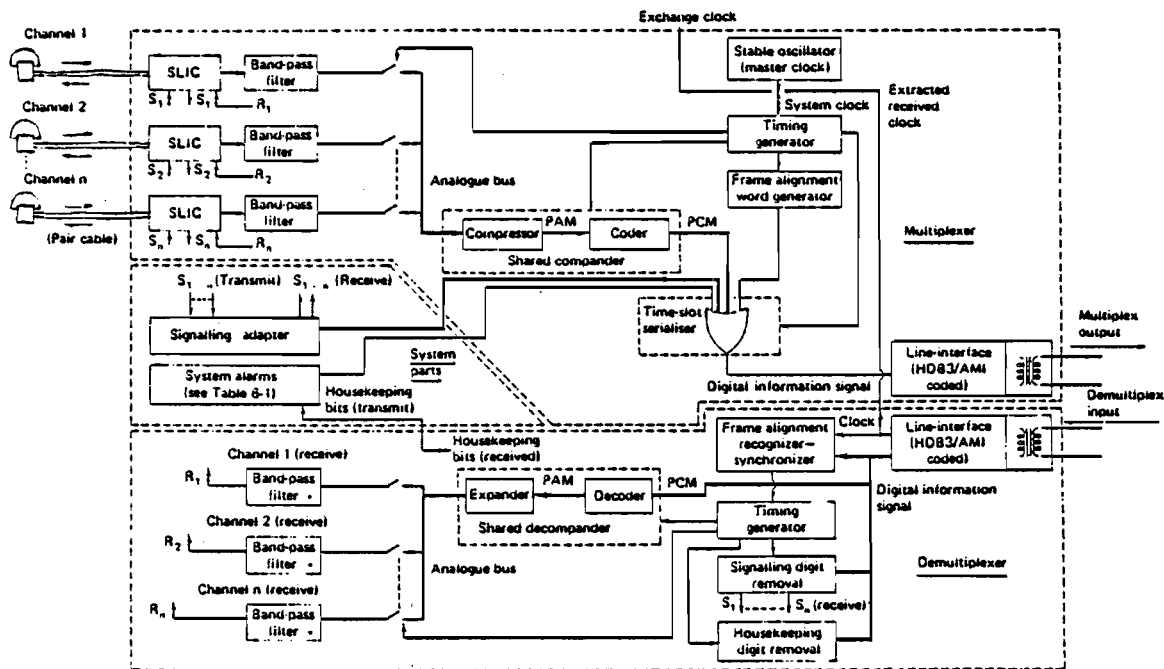


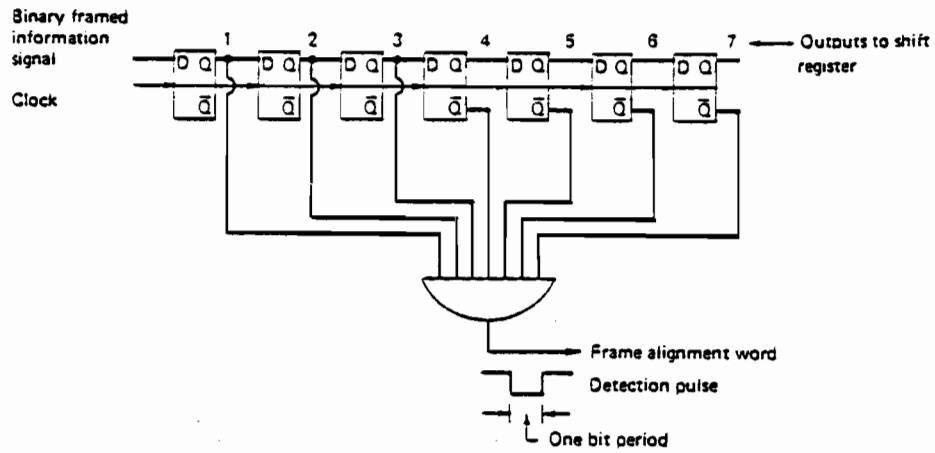
Figure 1-3. Example of TDM system complexity.
 (From PCM and Digital Transmission Systems by Frank F.E. Owen; Pg 132)

required for time synchronization and frame synchronization. The system also uses HDB3/AMI[†] line coding to give desirable transmission properties to the multiplexed signal.

TDM has problems with synchronization; problems that require sophisticated hardware for correction. Typically, the first problem to solve is how to provide time synchronization so the demultiplexer knows where to sample the incoming waveform. Secondly, the first time slot must be located to provide a frame reference position. A repetitive code the receiver can recognize and track is generated in the first time slot for the purpose of gaining frame alignment. A frame alignment word recognizer is shown in figure 1.4. When it detects the frame alignment word (FAW) the system assumes it has detected the start of a frame. It may not have though, because the FAW can appear in the data streams being multiplexed. The demultiplexer checks the next two or more frames to verify that the FAW is present. If it is after several frames, then alignment is considered achieved and demultiplexing can begin. The system continues to check the FAW each frame. Should it be absent for several frames do to corruption by bit errors then it assumes that alignment has been lost and begins the search process again.

Sending the frame and clock synchronization information requires additional complex hardware. Special line coding can be used to provide better clocking (as was done in Figure 1.3). Still, we can expect that

[†] The High Density Bipolar Codes (HDB_n) use the Alternate Mark Inversion (AMI) scheme except that the number of zeroes between ones is limited to n.



	Bits outside shift register				Shift register outputs								Recognition status
	1	2	3	4	1	2	3	4	5	6	7	-	
0 step	0	0	0	0	Pattern enters shift register								
0 step	0	0	0	0	Pattern not detected
1st step	1	0	0	0	0	Pattern not detected
2d step	1	1	0	0	0	0	Pattern not detected
3d step	1	1	1	0	0	0	0	Pattern not detected
4th step	.	1	1	1	0	0	0	0	Pattern not detected
5th step	.	.	1	1	1	0	0	0	0	.	.	.	Pattern not detected
6th step	.	.	.	1	1	1	0	0	0	0	.	.	Pattern not detected
7th step	1	1	1	0	0	0	0	.	Pattern detected
8th step	1	1	1	0	0	0	0	Pattern not detected

. - Represents a logical 1 or 0 (random information assumed)
 1 - Represents a logical 1
 0 - Represents a logical 0

Figure 1-4. Frame alignment search mechanism.

framing will slip every so often. Digital voice transmission can tolerate slips since the human ear will smooth out the errors that result. Computer data transmission cannot tolerate slips since the computer cannot smooth out the data as the ear does. To make matters worse the alignment process is statistical in nature since the FAW can appear randomly in the data before the system locks onto the true FAW; consequently the delay till frame lock is known only on the average. For multiplexing a single PCM voice channel and one high rate data channel the system of figure 1.3 seems too sophisticated, expensive and unreliable.

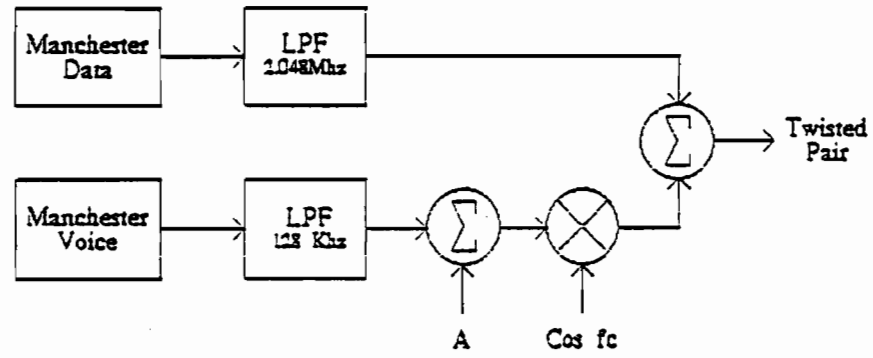
TDM/PCM suffers from hardware complexity. The other practical multiplexing scheme, frequency division multiplexing, suffers for similar, though less severe, reasons.

1.6.2 Frequency Division Multiplexing (FDM). A workable FDM scheme to accomplish the multiplexing task required by figure 1.1 is shown in figure 1.5.† This system is called voice over data. The idea is to translate the digital voice spectrum, using an analog carrier, above the spectrum (figure 1.6) required by the computer's data, then transmit the resulting signal.

Like the TDM/PCM system, both sources are manchester encoded to provide clock recovery (synchronization) information.

† Other modulations schemes, eg. PSK or FSK could be used for the voice however with the same problems as the simple AM scheme discussed here.

MULTIPLEXER



DEMULTIPLEXER

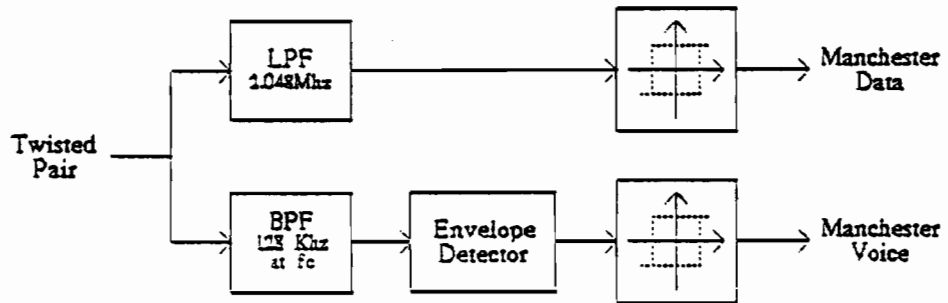


Figure 1-5. Voice Over Data (VOD) multiplexer and demultiplexer

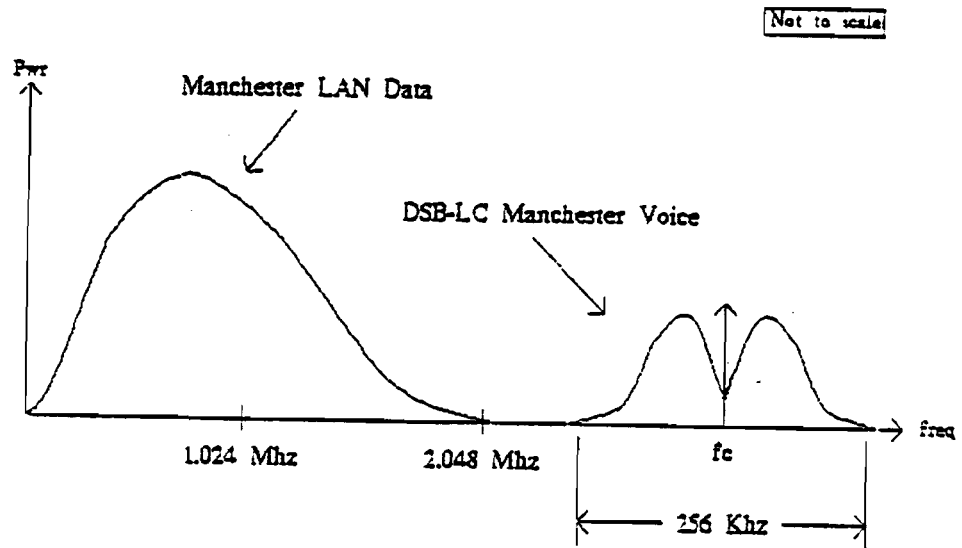


Figure 1-6. Spectral representation of VOD system.

Analog components dominate the FDM multiplexing scheme. Two low pass filters, two summers, and a multiplier are required in addition to a cosine signal generator. The demultiplexer requires two filters, high and low pass, plus an envelope detector to demodulate the double sideband large carrier (DSB-LC) signal carrying the digital voice. The demultiplexed signals are passed through schmitt triggers to regenerate binary signals.

The only difficult portion of this circuit to build is the multiplier located in the multiplexer. Since, only binary signals are being transmitted, a simpler implementation might be possible. A pure analog version will suffer from drift problems due to temperature, and will require adjustments to compensate.

The bandwidth required for operation (as shown in figure 1.6) is roughly as follows:

$$\begin{aligned} \text{BW} &= 2048 \text{ kHz} + 128 \text{ kHz} + 128 \text{ kHz} + \text{Guard bands} \\ &= 2300 \text{ kHz} \end{aligned}$$

2300 kHz is probably too high a bandwidth for the twisted pair cable to handle. The digital voice will be severely attenuated.

1.7 The rest of the story

The chapters that follow describe a new approach to multiplexing the digital voice and data channels. The scope of the text covers theory, simulations, and hardware implementations.

More specifically, here is how the chapter content is broken down.

Chapter 2 is describes the new baseband approach to multiplexing the voice and data. A theoretical description of the signal used is given and compared to TDM and QPSK.

Chapter 3 covers the design of the DVM system. The components of the DVM block diagram are discussed followed by signal characteristics and differential transmission operation.

The evaluation of the DVM system using simulation tools is completed in chapter 4.

Details of the DVM implementation in hardware and its performance are described in chapter 5.

Chapter 6 covers the application of the DVM approach using non-orthogonal sources at 160 KBPS and 1024 KBPS.

The conclusion and possible extensions to the DVM approach are discussed in chapter 7.

2. A NEW BASEBAND MULTIPLEXER APPROACH

2.1 Analog Sum Multiplexer

The first multiplexer design considered used the approach diagramed in figure 2.1. The 1024 KBPS manchester encoded data is added to the 64 KBPS manchester encoded digital voice.† Twisted pair wire transports the sum to the demultiplexer. High and low pass filters separate the data and voice signals respectively. Finally, schmitt triggers are used to reshape the signals to binary levels.

Figure 2.2 shows a time plot of typical waveforms and a spectrum of the sum. S1 is a manchester encoded sequence of sixteen bits. S2 is a zero bit manchester encoded. Also shown are the product and sum of S1 and S2. Integrating the product over one voice bit shows S1 and S2 are orthogonal.†† The spectrum shown in figure 2.2 shows two nearly distinct humps representing the data and voice spectrums. The use of the filters for demultiplexing is now apparent.

Simulations of the system shown in figure 2.1 operated as desired indicating that a hardware version should be attempted. However, this

† From here on the rate and the manchester encoding will be dropped. The 1024 KBPS manchester data will be called 'data' and the 64 KBPS digital voice will be called 'voice.'

†† The signals are orthogonal over one data bit since each data bit is plus one for half the bit and minus one for half the bit. Therefore the integral over one data bit is zero. The orthogonality is a result of the manchester encoding and the edge to edge alignment of S1 and S2.

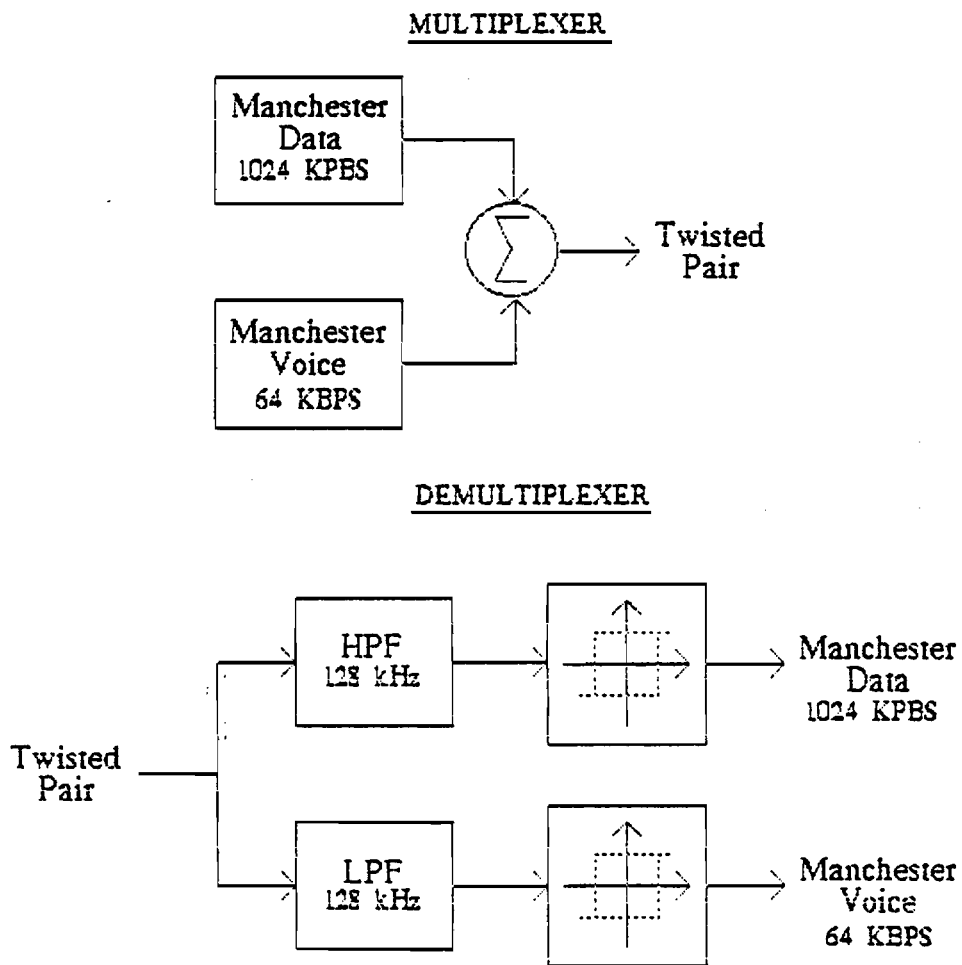


Figure 2-1. Analog sum multiplexer and demultiplexer.

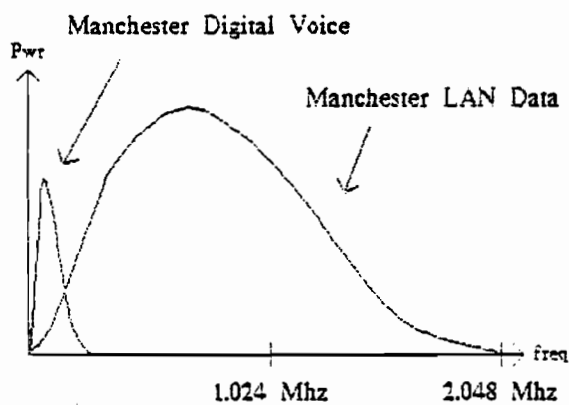
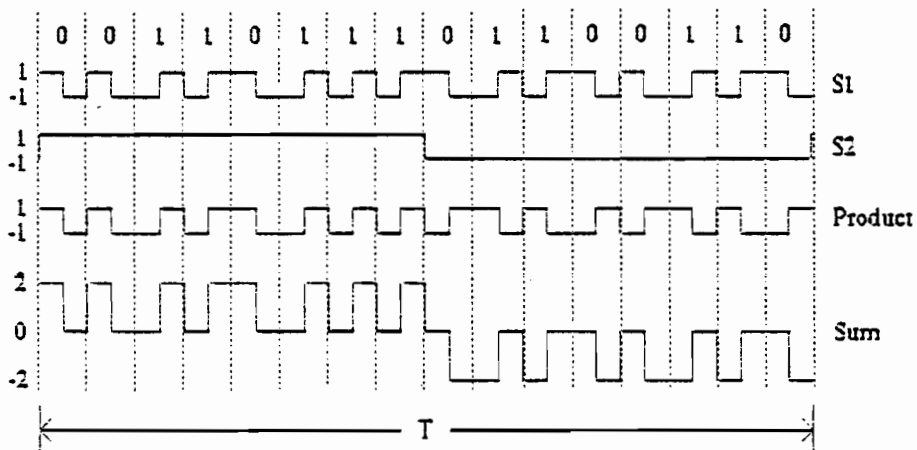


Figure 2-2. Analog sum signal and it's spectrum

scheme provided the basis for another approach that showed promise and lacked the analog filters with their associated problems. The new scheme is based on the product of the data and voice signals. It might be called a baseband quadrature phase shift keying (QPSK) system.

2.2 Analog Product

An alternate way to combine the signals is to multiply them together as shown in figure 2.3. The product is called the DVM (Data, Voice Multiplexer) signal and is similar to the sum signal shown in figure 2.2; it has a similar spectral shape and contains no DC (it is balanced). The data signal is defined with different amplitudes than the S1 signal in figure 2.2. The data signal takes on values of zero (0) and plus one (+1). Also, the integral of the product over a time T is equal to zero, so the signals are orthogonal.

The data is demultiplexed by using an absolute value function. Remember that the data signal is multiplied by a plus one, minus one signal; the absolute value removes the sign, leaving the data signal. Look at the product in figure 2.3 and mentally remove the sign, you can see that the data signal is recovered.

The voice signal is separated from the product by tracking the plus one, minus envelope present around the data. The schmitt trigger, shown in figure 2.4, performs this function. The data transitions cannot change the schmitt trigger output since they lack the amplitude to cross both (+1/2, -1/2) trigger points. A voice transition (+1 to -1, or -1 to +1) crosses through both trigger points causing a change in the schmitt trigger output.

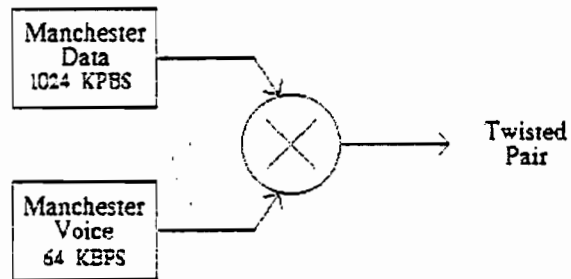
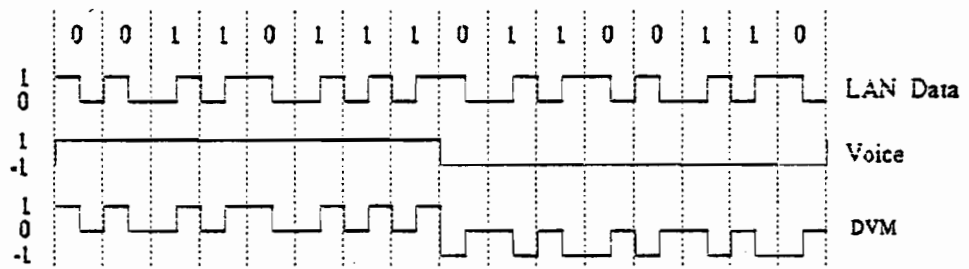


Figure 2-3. Analog product signal.

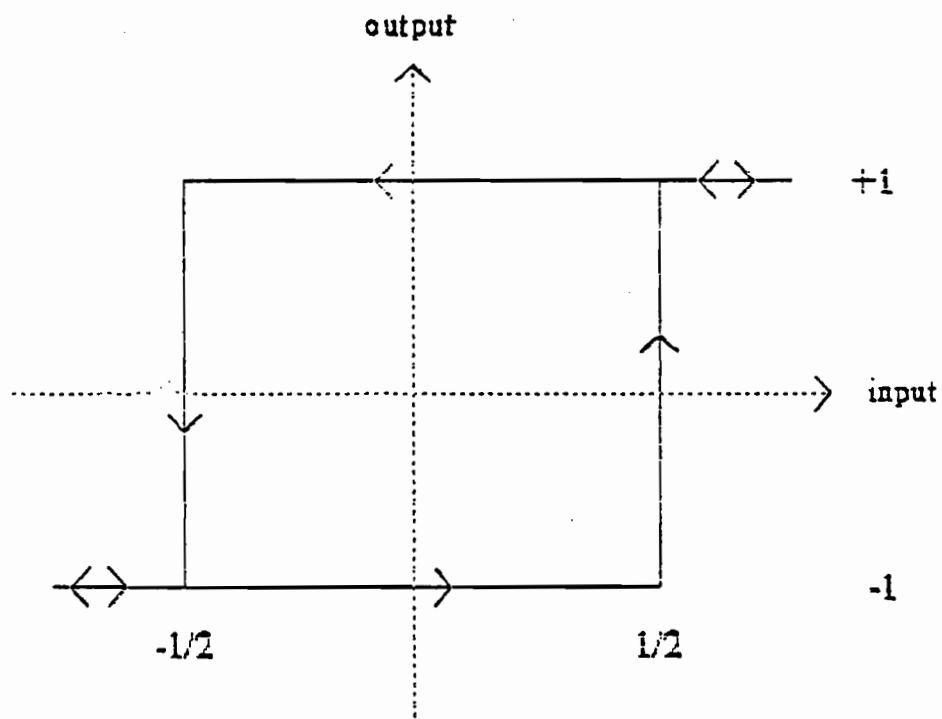


Figure 2-4. Schmitt trigger.

The product is not done with a conventional analog multiplier. Chapter 3 will cover the implementation and show a way to get the DVM signal from the difference of two signals derived using digital coding of the data and voice. To provide a common basis for evaluation of the proposed techniques a signal space representation of the DVM signal will be discussed.

2.3 DVM Signal Space Representation

2.3.1 Signal Space Concepts A signal space diagram consists of several (generally two, three at most) perpendicular axis'. [coop86] Each axis represents one orthonormal basis function from a set used to build the signals to be diagramed. A particular signal is represented by a linear combination of all basis functions. The coefficient of each basis function used to make a signal, collectively make an ordered n-tuple which is a coordinate of a point on the signal space diagram. Each signal gets a unique point assigned to it this way. With two basis functions and three values allowed for each coefficient, a set of eight signals are possible.

Figure 2.5 shows the basis functions ϕ_1 and ϕ_2 and the eight possible signals that can be created using plus one, zero, and minus one for coefficients. Note that ϕ_1 and ϕ_2 are the two possible high rate data signals. S_1 , S_2 , S_3 , and S_4 are the four component signals used to generate the DVM signal.

2.3.2 The TDM Signal Space S_5 , S_6 , S_7 and S_8 look like the signals that would be generated for a two bit time division multiplexing system,

with sources generating non-manchester encoded data with amplitudes of minus one and plus one. (A worthless approach to take since one bit would be needed for framing.) The point is in TDM, one source modulates phi 1 and the other source modulates phi 2. If additional basis functions (block pulses shifted successively further in time) are used we can see how a larger TDM frame can be constructed. The TDM signal components S5 and S7 lack a feature of S1 through S4: They lack any transitions. This makes clock recovery difficult (but not impossible.)

2.3.3 The DVM Signal Space S1 through S4 contain transitions in every data bit cell, maintaining the self clocking property of manchester encoded signals. The interesting aspect of the DVM signal components, and in contrast to the TDM components, is that the individual voice and data channels do not modulate phi 1 and phi 2 separately. Instead, phi 1 and phi 2 represent the two possible data channel signals and the voice signal determines the amplitude. The voice modulates both phi 1 and phi 2 independent of the data present in them.

The LAN data entering the multiplexer must be manchester encoded for this approach to work. Phi 1 and phi 2 will not be the signals shown in figure 2.5 if the incoming LAN data is not manchester encoded. It is a simple matter to manchester encode the LAN data so this is not a problem.

In addition, the signals S1 through S4 will only be generated if the manchester encoded sources are orthogonal. Introducing some phase shift between the signals will introduce some distortions in the product signal which is composed of various combinations of S1 through S4. If

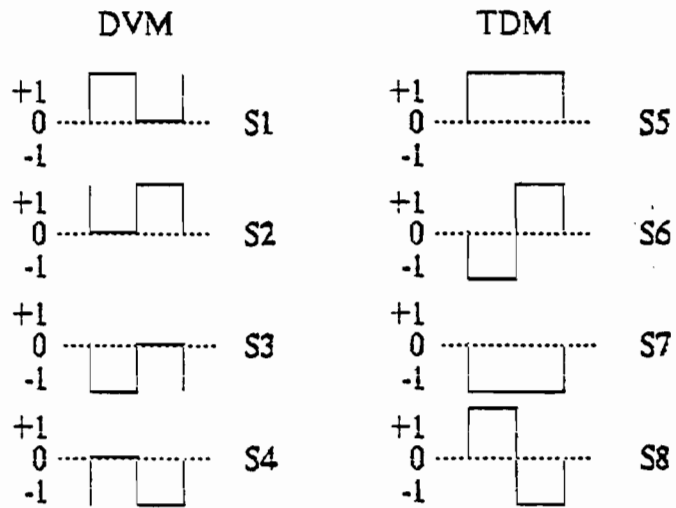
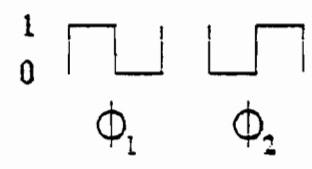
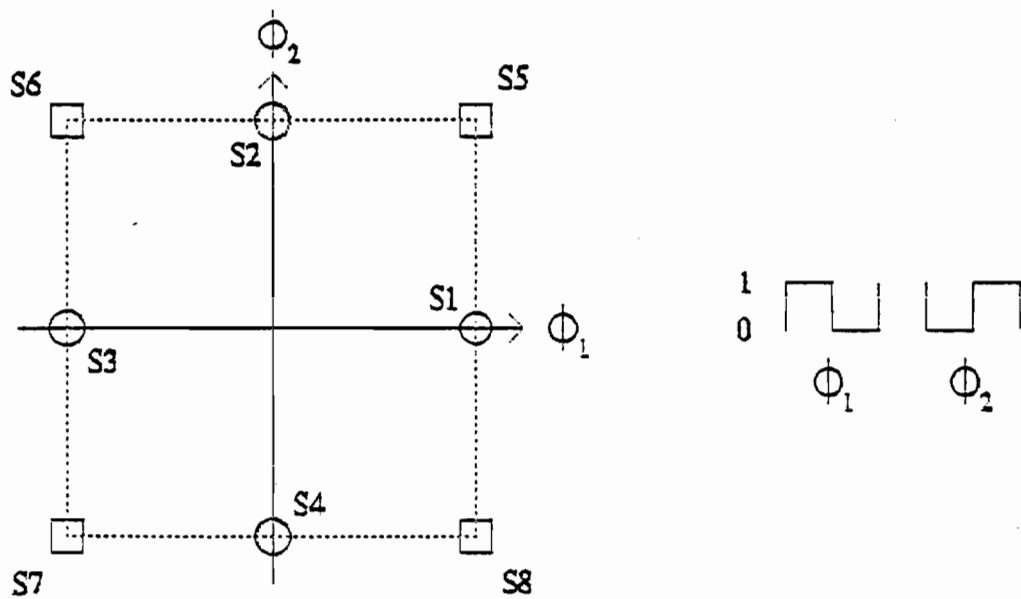


Figure 2-5. Signal space and signal components for DVM and TDM

phase shift is present the demultiplexing processes is not affected, but the processe of regenerating the individual data bits from the analog streams coming from the multiplexer is.

2.3.4 Quadrature Phase Shift Keying. Signal components S1 through S4 can be viewed as a form of QPSK; The signal S1 shifted by 90 degrees to form S2, 180 degrees to form S3, and 270 degrees to form s4. This is similar to the typical QPSK system which uses four phases of a sine waveform. The two channel TDM system can be block diagrammed (figure 2.6) in a similar way to the sine based QPSK system.

2.4 Baseband Implementation.

Chapter 3 covers the implementation of this scheme, primarily explaining how the two signals are multiplied together. Keep a finger on figure 2.3, the signals shown are useful for seeing how the system works.

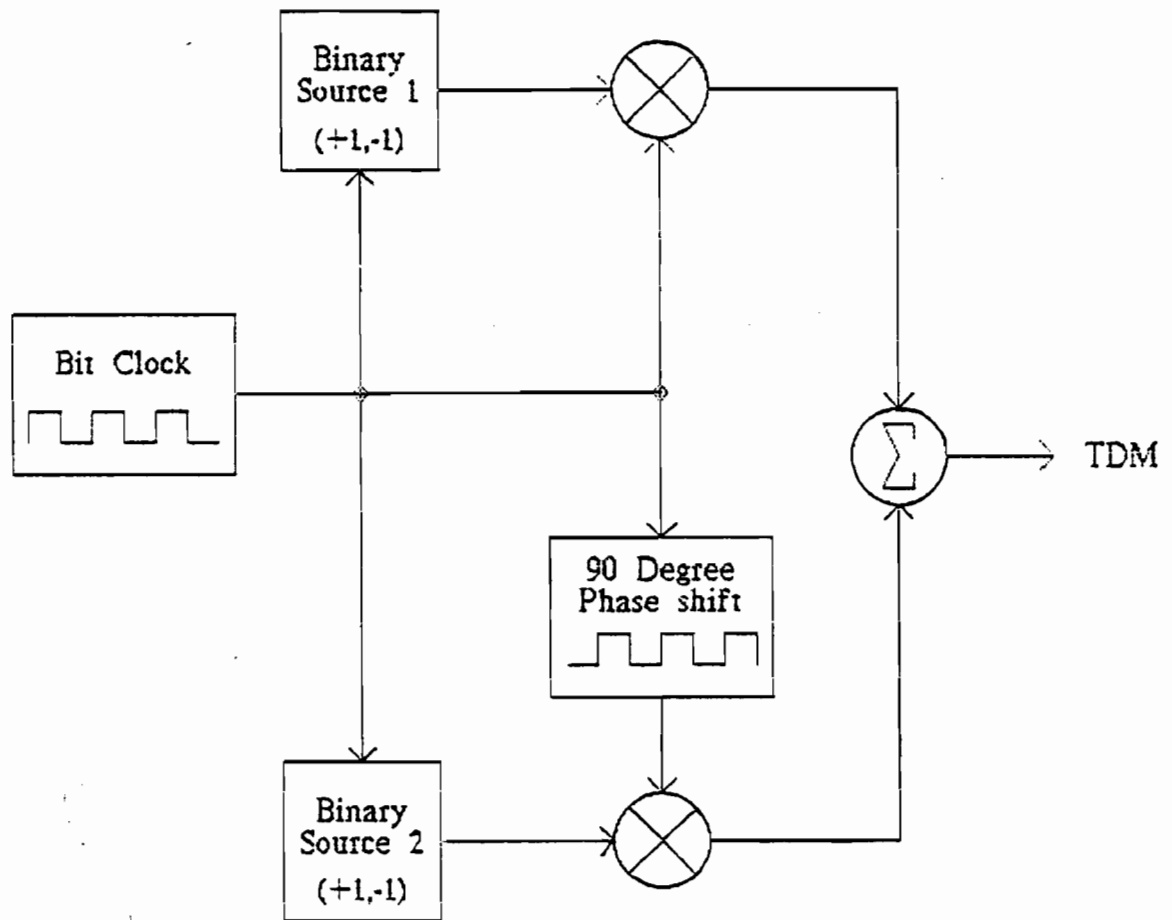


Figure 2-6. QPSK like implementation of TDM

3. Data Voice Multiplexer Design

3.1 Overview

This chapter describes the DVM system design. Greater detail is provided, particularly on how the multiply operation is performed. This chapter is the basis for discussions of the SYSTID [fash84] system simulations of the DVM (chapter 4) and the hardware design (chapter 5). The block diagram of figure 3.1 shows the top level view of the DVM. The individual blocks are discussed in order from the sources to the manchester decoders.

Remember that the research goal was the development of the multiplexer and the demultiplexer. The additional blocks (voice and data generators, channel model, and manchester decoders) are required for testing purposes.

The blocks in figures 3.2 through 3.6 are labeled in italics with the names of the corresponding simulation model names. Also, the simulation node names are included for easy reference to the simulation code that will be discussed in chapter 4.

3.2 Voice and Data sources

The sources are diagramed in figure 3.2. The sources are modeled as random NRZ binary waveforms with levels 0, +A. Figure 3.2 shows two clocks; one driving each RANPLS (random pulse) block. The clocks run at 1024 kHz and 64 kHz (also 160 kHz or 128 kHz), and are edge synchronized. On each rising edge of the clock, a new pulse level is generated with equal probability. This pulse is then manchester encoded

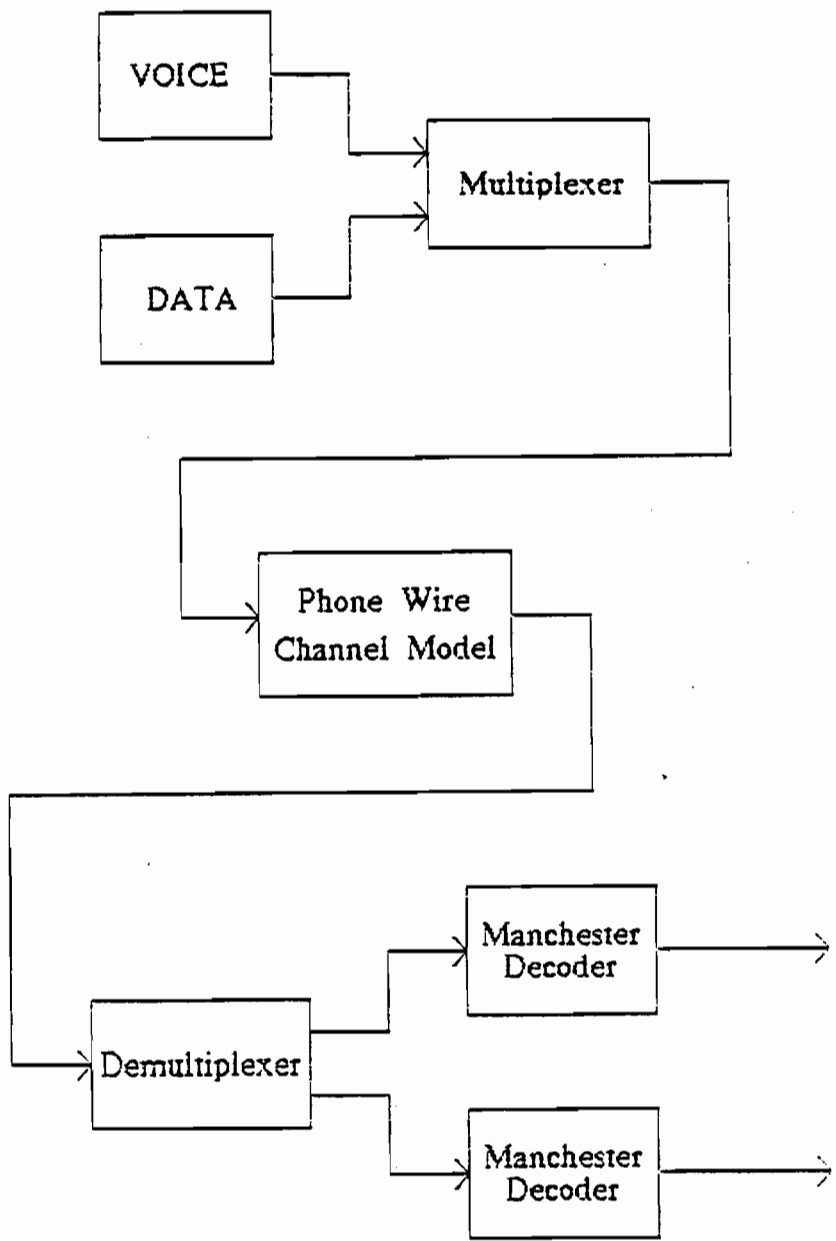


Figure 3-1. General DVM system organization.

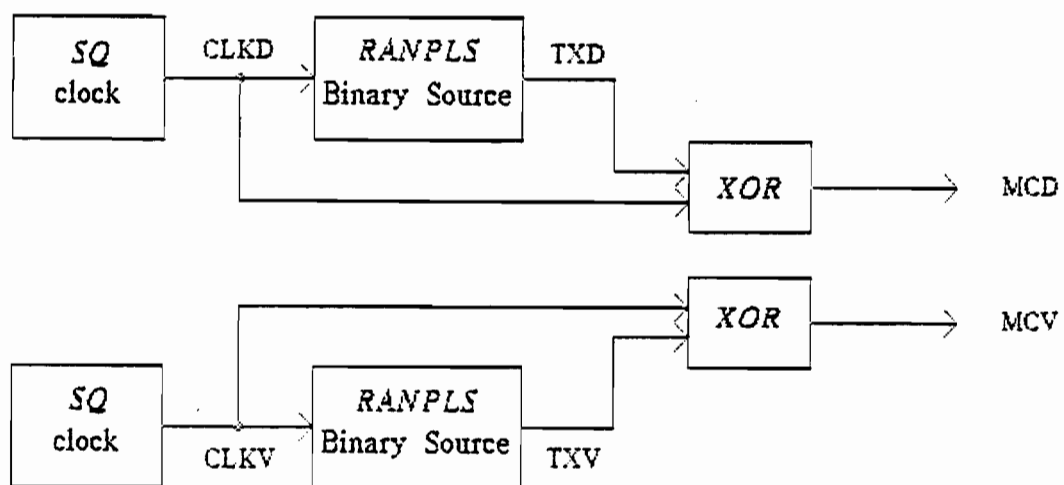


Figure 3-2. Manchester encoded, random binary sources.

by taking the exclusive-or of the clock signal and the binary waveform.

To see that this operation generates a manchester encoded waveform consider the following table:

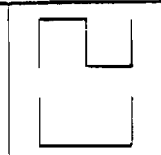
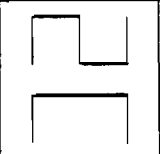




Clock		
Signal		
XOR		

TABLE 3-1. Demonstration of Manchester Encoding

3.3 Multiplexer Product Implementation

This section explains how the required product is efficiently performed. Recall from chapter 2 that the manchester encoded voice and manchester encoded data are multiplied together. The two signals were not defined the same so it would seem that the multiply would have to be done with a pure analog multiplier. Fortunately, this is not the case. The capability to multiply by plus one (+1) and minus one (-1) is all we need.

The product is formed using an unusual combination of digital and analog components. † Table 3.2 and figure 3.4 illustrate how the product

† Keep in mind that digital hardware uses voltages to stand for logical true and false values. At one point the voltages representing true and false are subtracted in analog fashion. Do not let it scare you!

is obtained. In table 3.2, starting at the left, the four possible combinations of the binary waveforms MCV (manchester coded voice) and MCD (manchester coded data) are shown. MCV and MCD are the inputs to the DVM system. Internal to the multiplexer block shown in figure 3.3, the two signals drive 4 to 1 digital multiplexers (mux) which act as code lookup tables. The four possible combinations of MCD and MCV select one of four values to output for each 4-1 mux. The PLUS and MINUS columns of table 3.2 correspond to the code values of the two 4-1 mux's. The difference column shows the result of an analog subtraction of the voltages representing the digital 1's and 0's. This difference is the product formed by multiplying the voice and data signals defined in figure 2.3! Thus showing that figure 3.3 will generate the DVM signal. An interesting feature of this implementation is that a difference of two signals is involved. The difference operation can be placed in the receiver and noise can be removed by just subtracting it out!

MCV	MCD	PLUS	MINUS	DIFFERENCE	PRODUCT	VOICE	DATA
0	0	0	0	→ 0	0	-1	0
0	1	0	1	→ -1	-1	-1	1
1	0	0	0	→ 0	0	+1	0
1	1	1	0	→ +1	+1	+1	1



TABLE 3-2. DVM signal product formation.

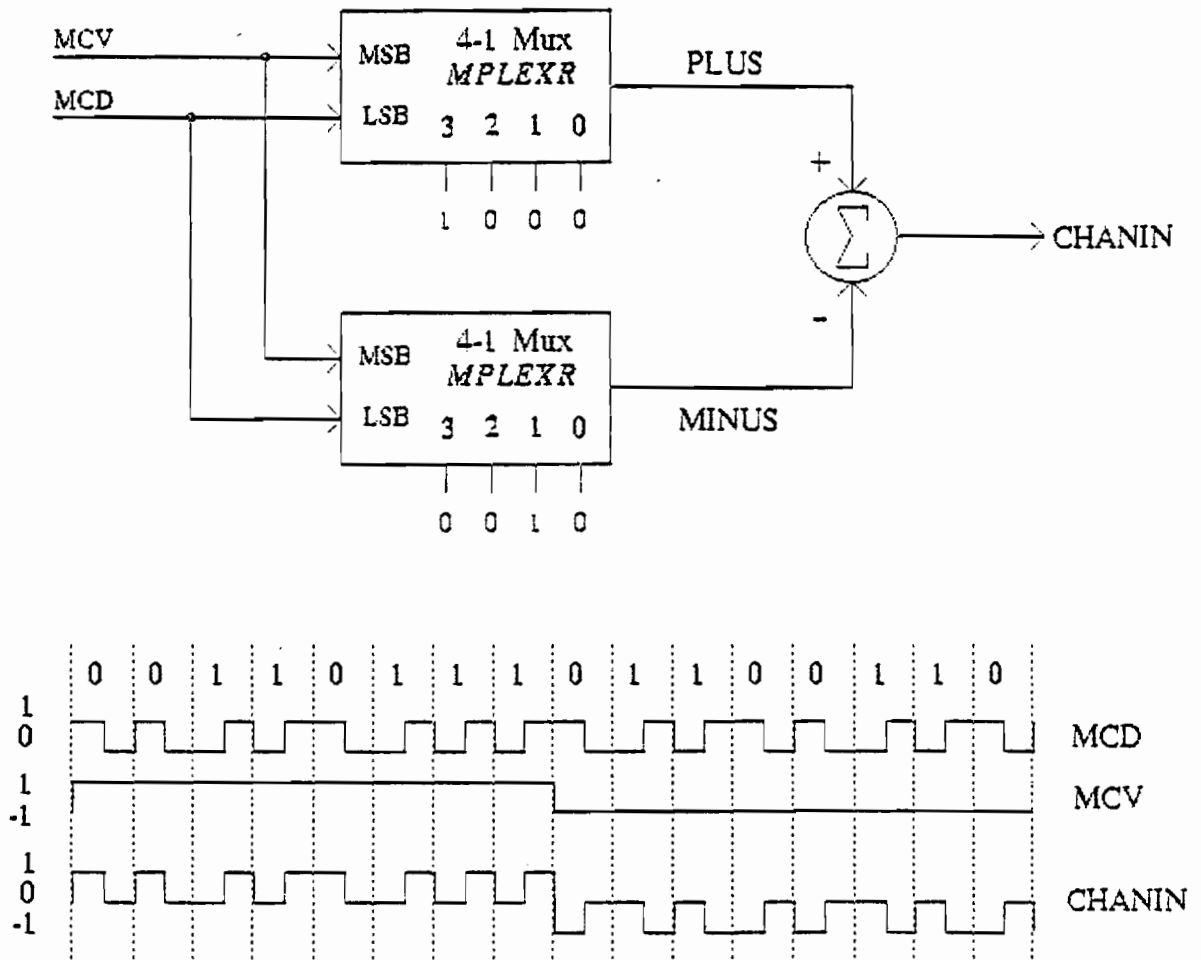


Figure 3-3. DVM multiplexer implementation

An alternate method to obtain the product is to use the approach shown in figure 3.4. Here, the MCD signal is added to, or subtracted from, zero under the control of MCV. When MCV is plus one (high), MCD is added to zero (multiplied by +1). When MCV is zero (low), MCD is subtracted from zero (multiplied by -1). In this way, MCV's sign is transferred to MCD; exactly what happens when MCV and MCD are multiplied together. The digital implementation is considerably simpler than any analog multiplier. There are no components to adjust and no problems with drift.

The greatest advantage with this implementation is where the differential amplifier can be placed. The obvious placement (in the receiver) makes a tremendous impact on the amount of noise the system will handle.

3.4 The Channel.

The channel model is covered in detail in chapter 4. Briefly, a frequency domain transfer function for the channel is generated, then an FFT (Fast Fourier Transform) is taken to find the impulse response. The output of the channel is obtained by convolving the input with the impulse response. It then passes on to the demultiplexing section.

3.5 The DVM demultiplexer

The demultiplexing operation (figure 3.5) consists of two simple functions. The three level DVM signal (figure 3.3) is sent to both functions and the respective signals are pulled out.

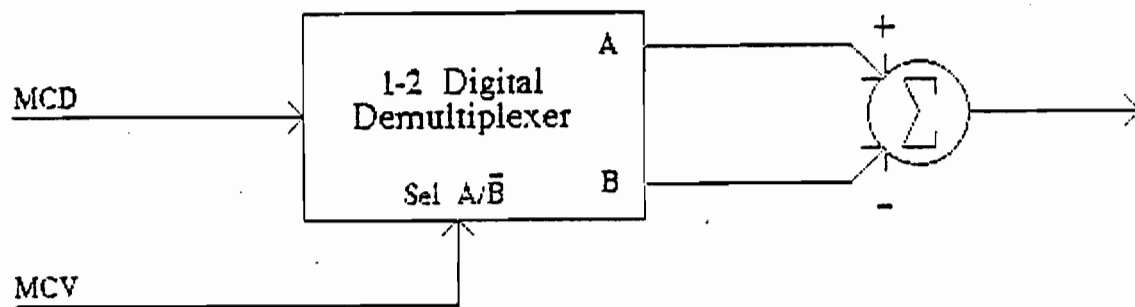


Figure 3-4. Alternate DVM multiplexer implementation.

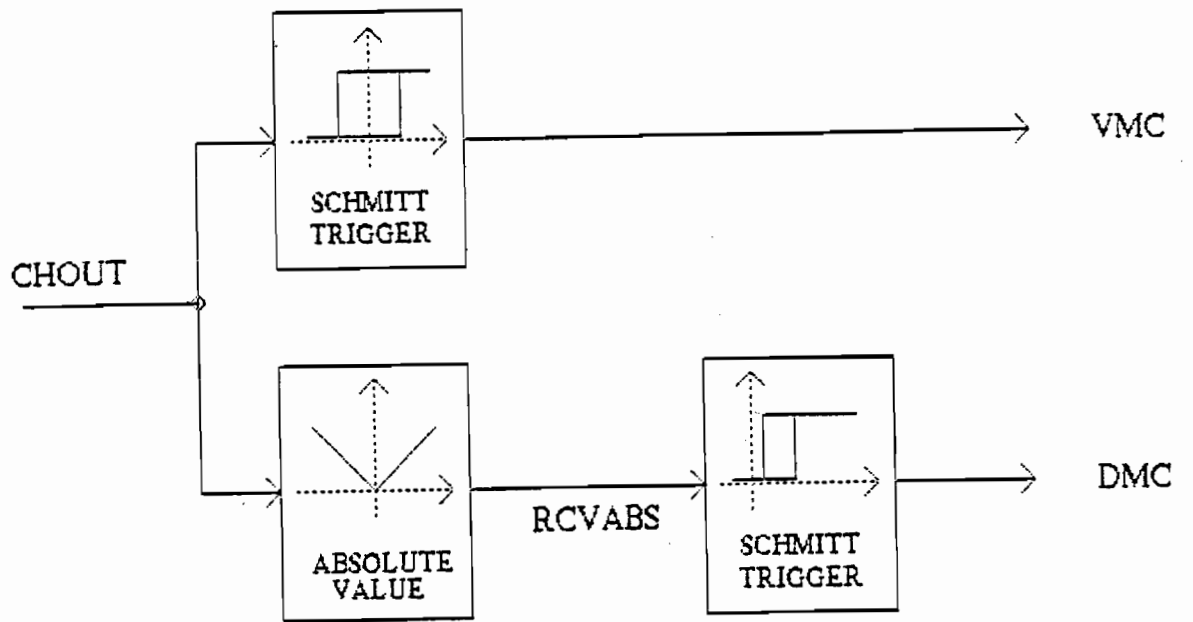


Figure 3-5. DVM Demultiplexer

The manchester encoded voice is recovered by a schmitt trigger with trigger levels at $-A$ and $+A$. This choice of levels results in the output of the schmitt trigger being the sign of the input signal.

The manchester encoded data is recovered by ignoring the sign of the three level signal. An absolute value function performs this task.

Figure 3.5 shows an additional schmitt trigger in the data recovery subsystem. The absolute value output is an analog signal that, after going through the channel, will have rounded edges and slow rise times. The binary signal is recovered by the schmitt trigger. A simple threshold could have been used but for various reasons discussed later the hardware was implemented with a TTL (transistor transistor logic) family schmitt trigger. To make the simulations as close to the hardware as possible the schmitt trigger was modeled as a 741s14 TTL gate.

3.6 Manchester decoders

The final blocks of figure 3.1 to cover are the manchester decoders. The manchester decoder outputs NRZ (non return to zero) data. Its operation is complex compared to the simple exclusive-or gate used for manchester encoding.

The decoding process, shown in figure 3.6, begins with the generation of a delayed version of the input signal MC. MCDLY, the delayed MC, is then exclusive-or'ed with MC. The resulting signal called TRIG, consists of narrow pulses at the location of every rising and falling edge of MC. The center of every manchester encoded bit contains a rising or falling transition; transitions at the start or end of any cell is not guaranteed. A "filter" is needed to remove all the

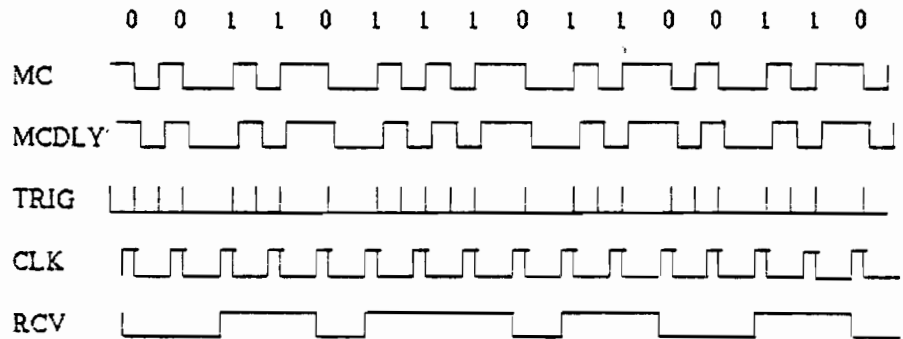
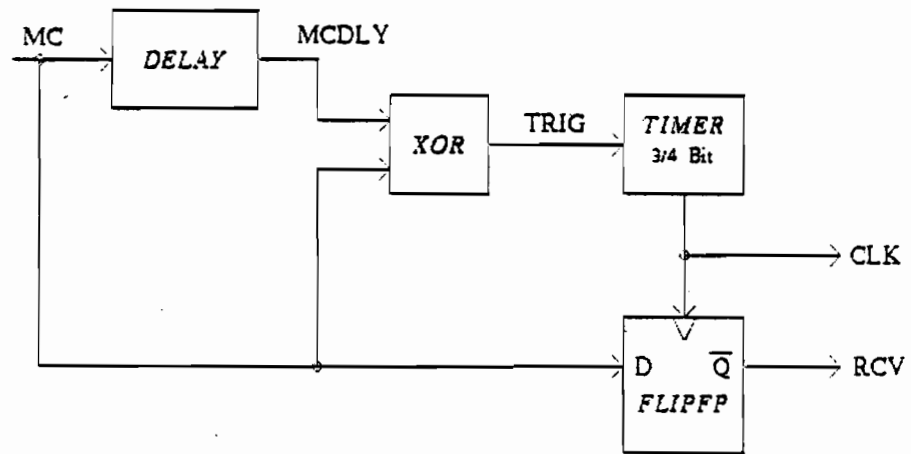


Figure 3-6. Manchester decoders.

start and end cell transitions and keep the center cell transitions. The resulting signal can be used as a clock to recover the NRZ data.

The filter is made with a timer circuit. When a TRIG pulse occurs the timer is started for a time equal to three quarters ($3/4$) of the period of the manchester encoded data. For the 1024 KBPS data stream this is 732 ns. The timer cannot be retriggered until it has turned off again.† The combination of these two properties causes all the leading and trailing cell pulses to be discarded. Once the timer is synchronized, the $3/4$ bit delay will always skip over any pulses in between the center cell pulses. The timer gets synchronized when MC changes from a zero to one or one to zero bit cell. The output of the timer becomes the CLK signal which clocks the D-Flip flop to sample the incoming MC. It will always sample in the first half of the cell. If it samples a high value it knows the cell represents a zero, a low value represents a one. The NRZ data is taken from the Q bar output of the flip flop for this reason.

Next the signal properties of the DVM signal are covered.

3.7 DVM signal properties

The DVM signal has two desirable properties. First, every cell contains timing information since a transition occurs in the center. The manchester decoders use this center transition for to regenerate the sample clock. Second, the signal contains no significant DC level.

† The timer output goes HIGH when it turns OFF.

This allows the signal to be transmitted through transformers. (Transformers will not pass DC.) This is significant because with transformer coupling, power can be supplied to the circuit over the same wires used to transmit the signal.

Having developed the necessary background for understanding the theory and operation of the data/voice multiplexer we can move on to the simulation and hardware design of the system discussed next.

4. SIMULATING THE DATA AND VOICE MULTIPLEXER SYSTEM

4.1 Simulation Purpose

The first simulation program developed for the DVM had one primary use. It served as a testbed for finding out quickly whether the DVM would work with a low pass channel. Once the simulation prototype worked, hardware construction began.

Several questions arose from the initial simulation prototype. Two of them were initially focused on: The first was to find out how well the system worked when the two data sources were not driven by a common clock. The second was to find out the system performance as a function of distance.

The nature of hardware makes these questions difficult to answer. To answer the first question using the hardware would require the hardware to have a way to generate a controllable phase shift between two clocks in a continuous manner. The second question would require a large amount of wire to be chopped up until the system started working; An expensive solution.

The simulation design and organization is shaped around these questions and around problems uncovered through experimentation with the simulation model.

The sections of the chapter that follow describe the organization, design and results of simulation models developed for the DVM. Section 4.2 describes the simulation organization through the use of an overall system block diagram. Section 4.3 explains the basic design of the

simulation components so that the simulation results of section 4.4 can be easily explained and understood.

4.2 Simulation Organization

Each subsystem of the DVM has a corresponding simulation model as shown in figure 4.1. Simulations are written in the SYSTID simulation language [fash83]. Node names are given where signals are used to connect inputs and outputs of the models. SYSTID allows for a hierarchical organization of simulation models. A few of the subsystems are coded into models, allowing the subsystems to be used several times, while others are simply a few lines of code.

The simulation mainline consists of five major parts. Several models together constitute the data and voice random bit generators and manchester encoders. Following this is the multiplexer. The multiplexer feeds the channel model which outputs to the demultiplexer. The last few modules recover the NRZ bit streams from the multiplexed manchester encoded data and perform bit error detection.

The philosophy has been to match the hardware and simulation prototypes as much as possible. The closest match is between the functional realization of the multiplexer and demultiplexer. Other portions differ because of difficulties in simulating the exact operation of the hardware, or because the question to be answered required difficult hardware modifications. The testing portions of the both hardware and simulation systems allow the most flexibility in modeling so they are the least similar. The next section covers how the simulation components were implemented and highlights the places where

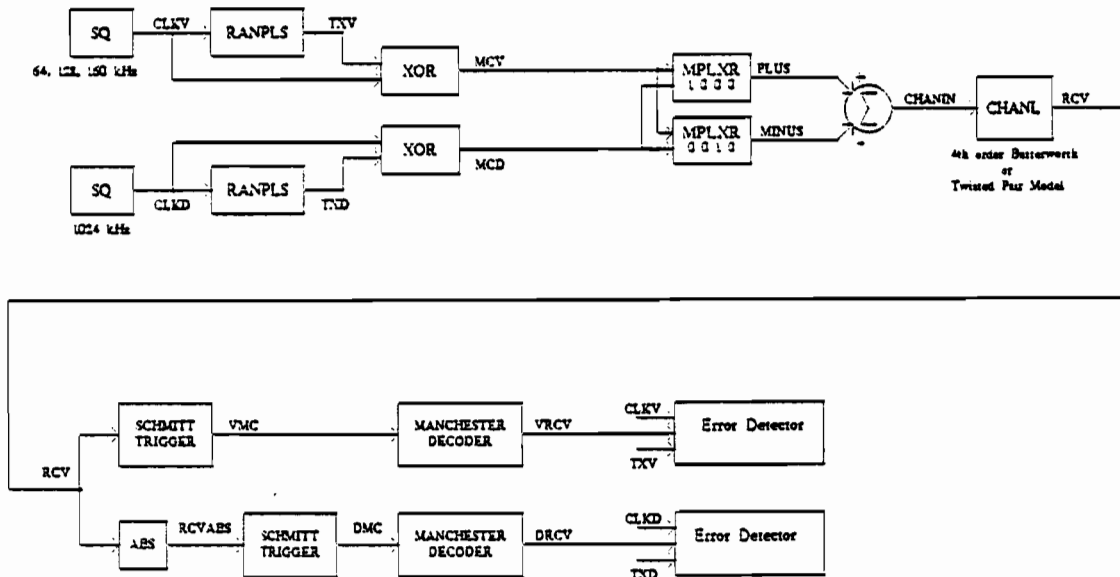


Figure 4-1. Simulation system block diagram.

hardware and simulation differ.

4.3 Simulation Design

This section covers the essential modeling details of the various components in the simulation. The first section covers analog components including the channel and schmitt triggers. The second section covers the digital components. The last section covers the error detection scheme.

4.3.1 Analog component modeling. Two types of channel models were used in the course of simulating the DVM system. The first model used to characterize the channel was a fourth order Butterworth low pass filter (supplied by SYSTID). The second model is meant to approximate the actual channel more precisely. It models a transmit side transformer followed by twisted pair, followed by a receive side transformer.

Figure 4.2 show the process used to obtain the channel model. First the lumped constant parameters (resistance, inductance, conductance and capacitance) of the twisted pair plus the length are used to define a frequency domain transfer function for the wire. The R, L, G, and C parameters are frequency and cable gauge dependent and are generated from functions found in [bell77]. The transmission line function is valid only when an infinite line or a line terminated in it's characteristic impedance is used. The hardware will have proper termination so this is reasonable.

The transformer's frequency domain characteristics are generated next. Each transformer is modeled as a second order, bandpass,

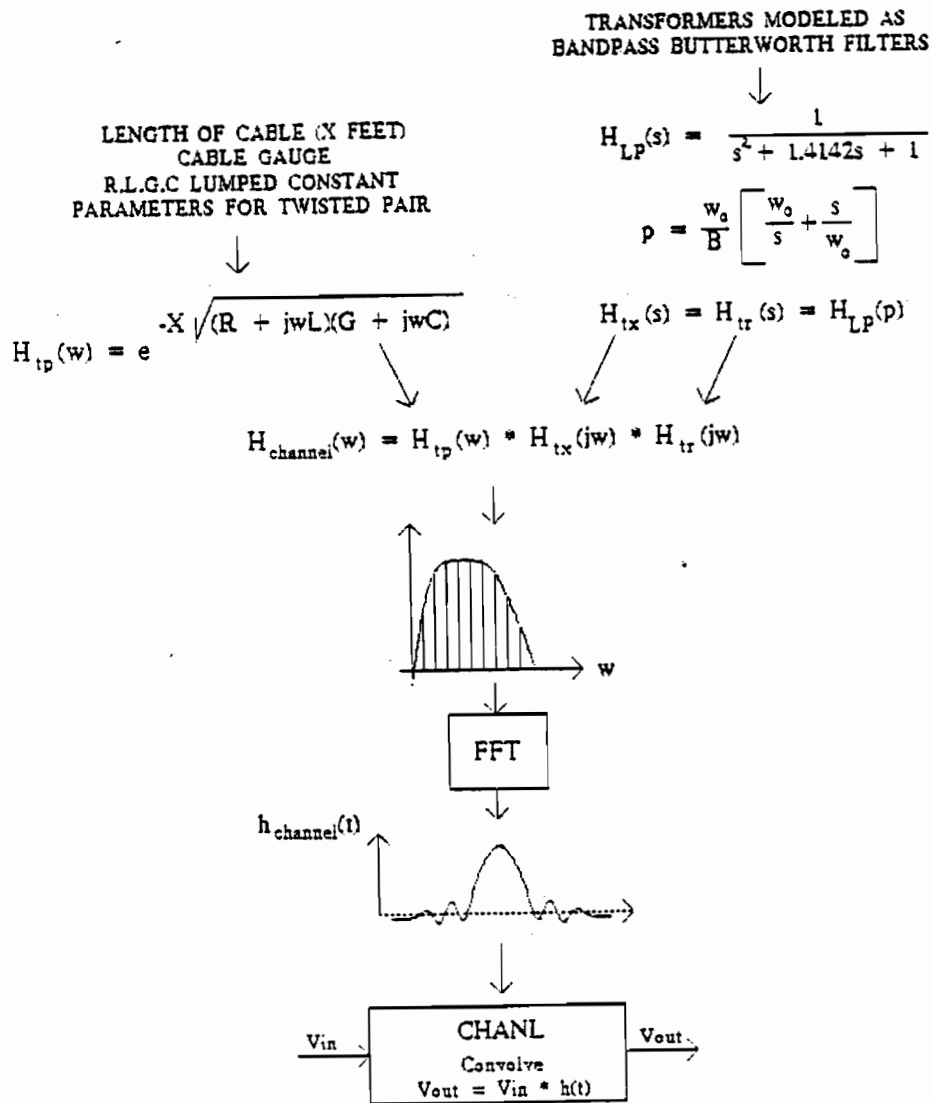


Figure 4-2. Twisted pair modeling process.

butterworth filter. The overall channel frequency response $H(f)$ is the product of these functions.

Next, $H(f)$ is sampled at 4096 points and an FFT (Fast Fourier Transform) is done to find the impulse response $h(t)$. The energy in the tail of the impulse response energy is trimmed back a few percent to reduce the number of samples that will be used by the tapped delay line. The CHANL model reads the trimmed impulse response from a file and sets the taps on a tapped delay line to convolve the input with $h(t)$.

The other major analog component, the schmitt trigger, is half analog. The schmitt trigger accepts an analog input and regenerates a binary signal from it using hysteresis instead of simple level comparison. The SYSTID model allows for selectable high and low trigger and output levels. This is primarily how the digital components in the next section are modeled.

4.3.2 Digital Component Modeling. The digital components have simplified models. In particular no attempt is made to model propagation delays. This is reasonable, since only a few digital components are used the propagation delays do not accumulate to significant amounts. Rise and fall times of the waveforms are ignored and perfect square pulses are used. The bandwidth of the digital devices is so much greater than the channel that the channel is primarily responsible for increasing rise and fall times in the transmitted signal. Lastly, the output and reference levels are selected to imitate TTL logic devices. The simulations use values of 4.5, 0. and 1.5 for high, low and reference respectively.

The implementation of the four components used, D flip flop, 4 to 1 multiplexer, exclusive or gate, and timer are found in their simulation code documentation comments (the simulation code is in appendix A).

Some of the digital devices are used in the error detection system described next.

4.3.3 Error detection subsystem. The error detection system compares the original transmitted binary data with the received binary data using an exclusive or gate (XOR). The XOR output is sampled by the falling edge of the transmit clock. If the transmit and receive streams differ, then a counter is incremented and the time of the error is stored. This information is printed at the end of the simulation run. Since the delay between transmitted and received signals did not exceed one half bit time in the hardware, it is ok to use the transmit clock falling edge to sample the XOR of the undelayed TX and RX signals.

Next the performance of the simulated DVM is examined, primarily through the use of eye diagrams and the output of the error counters on the data and voice channels.

4.4 Simulation Results

The simulations were done primarily with two DVM programs. The programs are called DAVIS06 and DAVIS07. They are nearly identical except DAVIS06 uses a 4th order Butterworth filter for a channel and DAVIS07 uses the twisted pair model described earlier.

Three investigations are described in the following sub-sections: Performance of DVM using Butterworth channel and twisted pair models and

finally clock skew effects are examined.

4.4.1 System Performance with a Butterworth Filter Channel Model. To answer the question about feasibility, the DVM was simulated with a fourth order Butterworth low pass filter for a channel. (The SYSTID model {davis06} for this system is found in appendix B.) To judge whether the DVM worked, two criteria were used: The error detectors had to register zero errors and the eye diagrams of the channel output (RCV) had to "look good" (i.e. look decodeable). The DVM performance capability is more easily judged with eye diagrams than with error counters. Since ISI (Inter Symbol Interference) is the primary limiting factor in system performance (noise is not a factor since the differential operation in the receiver is going to remove most of it) the error counters tend to give a binary result: The system works or the system does not work. This is not particularly useful since the error counters are telling information about how well the manchester to NRZ data decoders are working and not how well the demultiplexing process is working. All though more subjective, the eye diagram is more useful so it will be used throughout the remainder of the chapter to answer DVM system analysis questions.

There are several features of the data visible in the eye diagrams. These features are shown ideally in figure 4.3. The most prominent is in the middle of the diagram where the guaranteed edge in the center of every bit cell occurs. One quarter of diagram off from the center is the start/end of a bit. Here, the transition does not always occur and we see level portions as well as transitions. The last transition to be

seen is one caused by a voice bit changing from a zero to one or one to zero. This change shows as a trace going from full negative to full positive or vice versa. These diagrams show information primarily about the data signal since the choice for the window width is only two data bits. Note the low and high frequency envelopes caused by the alternating pattern {1010101...} and by the repetitive pattern {111...0000...}. The alternating pattern gives a 512 kHz signal and the repetitive pattern gives a 1024 kHz signal.

Figures 4.4 through 4.7 show eye diagrams of RCV (channel output) for channels with bandwidths 0.5 Mhz, 1.0 Mhz, 1.5 Mhz and 3.0 Mhz respectively. The diagrams are drawn with the symbol width equal to two bits (ie $1/512e3$ seconds). Also note that the y-axis scales differ from one figure to the next.

Figure 4.4 shows a completely closed eye from the repeating bit pattern. Since this pattern represents a 1024 kHz square wave it is not surprising to see it completely obliterated by the 0.5 Mhz bandwidth of the channel. As the bandwidth is increased (figures 4.5 - 4.7) the eye opens up enough that it can be decoded. The error counters indicated that the manchester decoding system cannot decode the recovered manchester data bit stream when the bandwidth was reduced below 2.75 MHz.

The other critical feature to extract from the figures is the jitter shown at the center of the diagram in the guaranteed transitions. The 1.5 MHz and 3.0 MHz show the transitions crossing at the same voltage level. However, the 1.0 MHz chanel shows appreciable distortion. The distortion will cause jitter in the decoder sampling

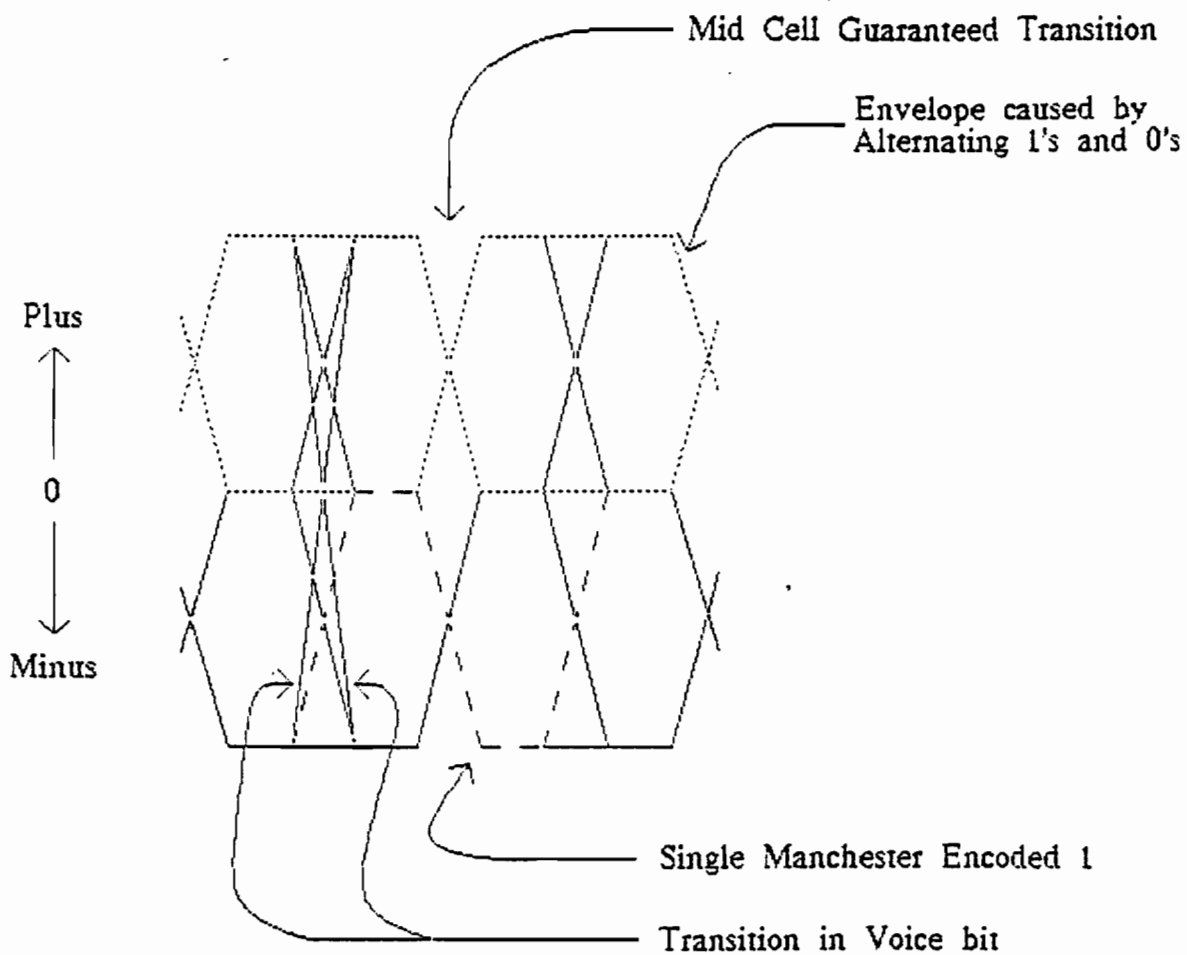


Figure 4-3. Prominent features in the eye diagrams.

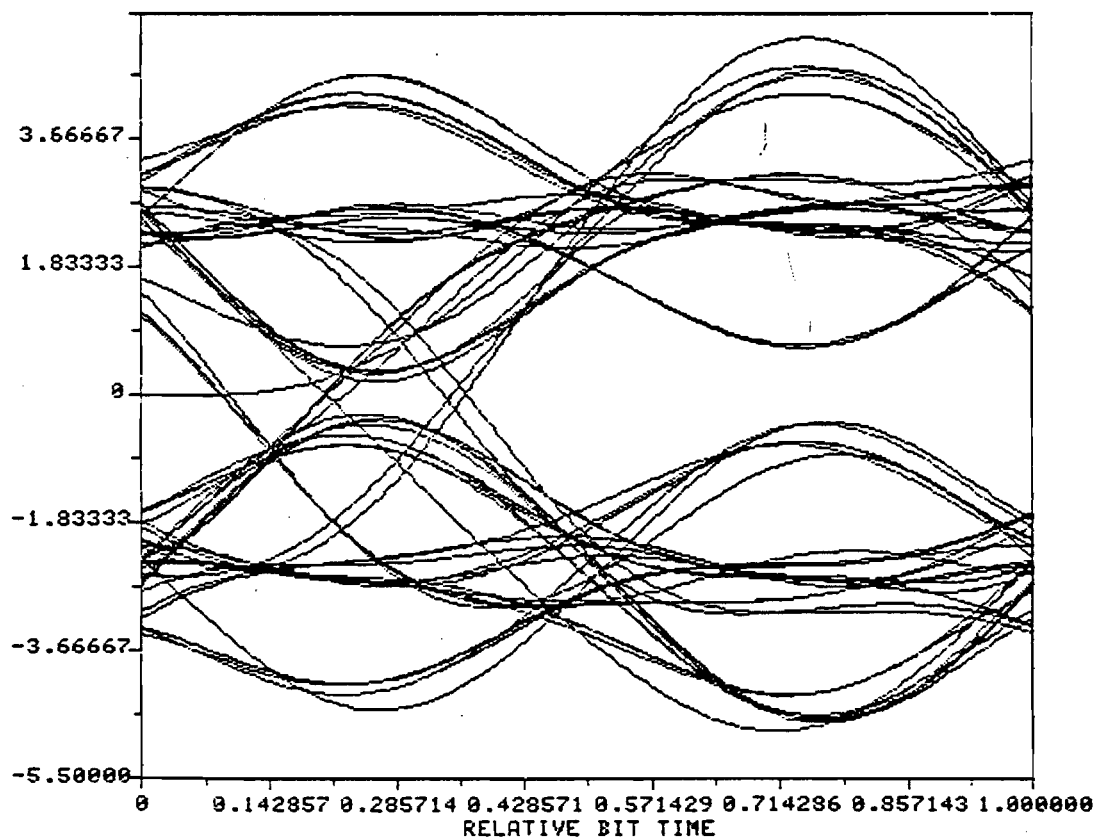


Figure 4-4. 0.5 MHz eye diagram.

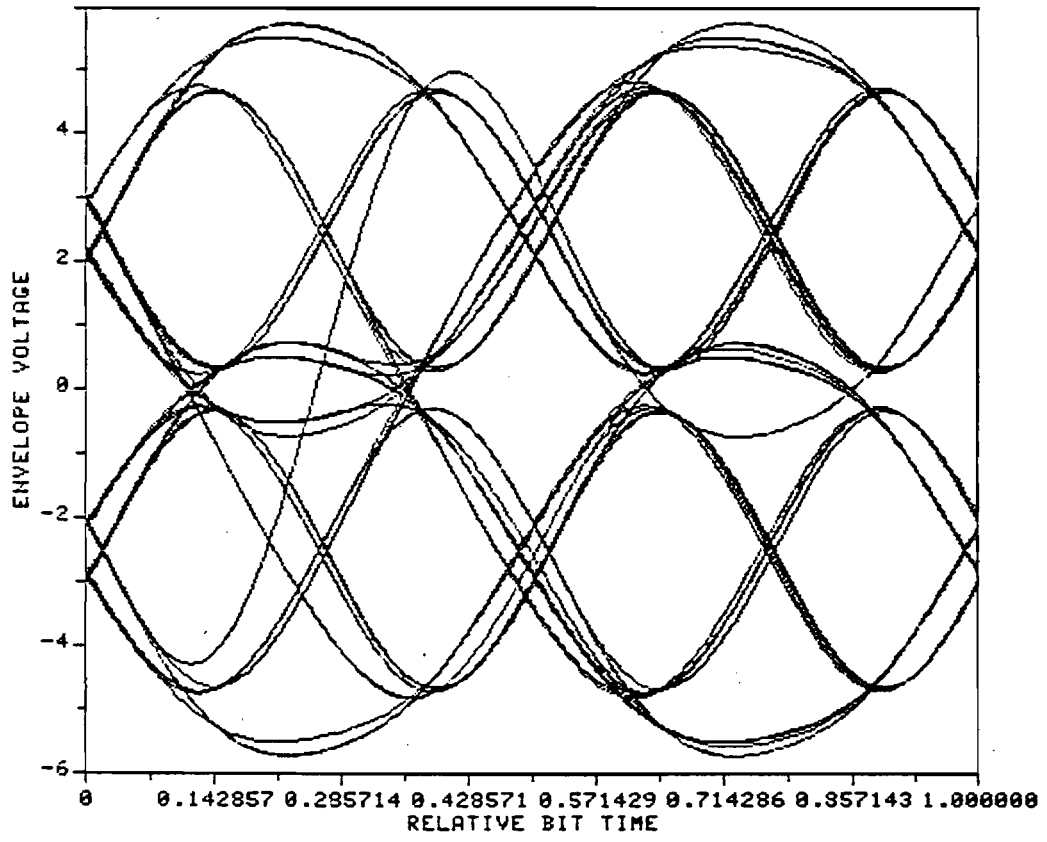


Figure 4-5. 1.0 MHz eye diagram.

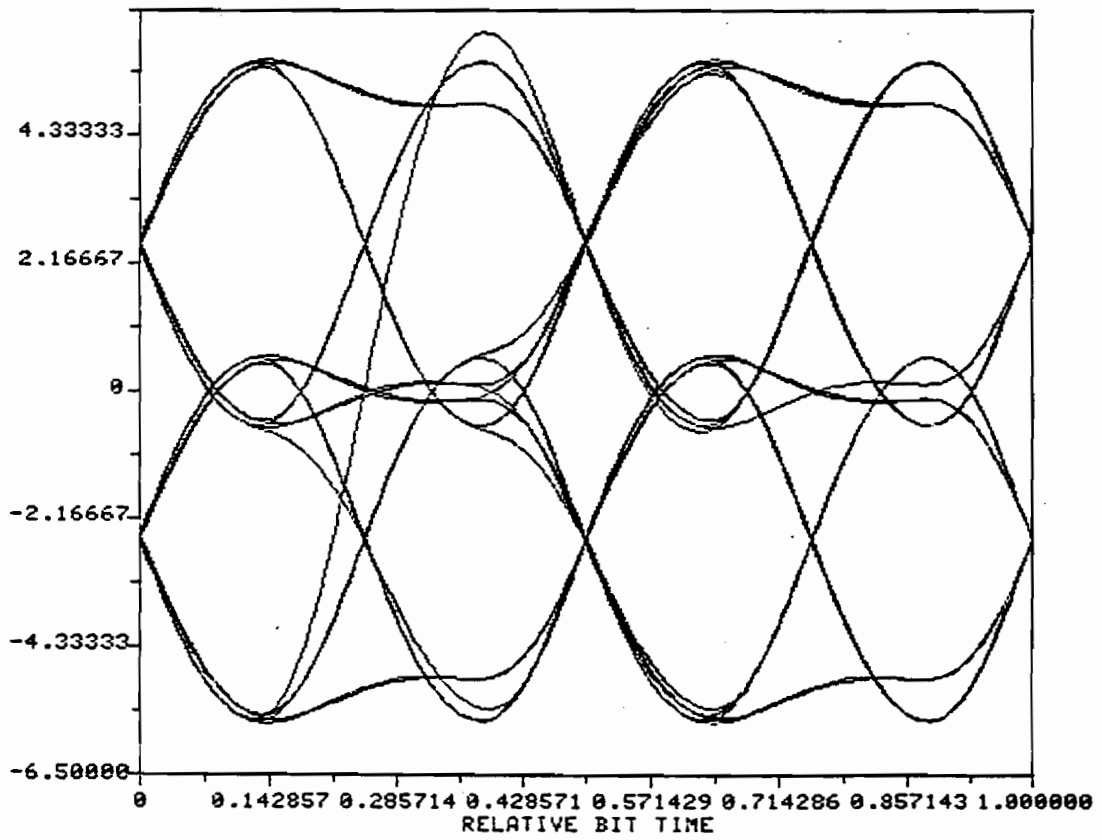


Figure 4-6. 1.5 MHz eye diagram.

clock since it is these transitions which the system extracts to recover timing information. The voice channel has jitter introduced because a voice bit transition might have an envelope represented by a 01 data bit pattern at the transition.

The next section covers the result from simulating with a twisted pair channel model.

4.4.2 System Performance with a Twisted Pair Channel. Eye diagrams are used again for a subjective determination of the system performance. The simulations use a more realistic model of a real twisted pair cable. The model takes into account the frequency dependence of the lumped constant parameters defining the cable. This is necessary since the DVM system uses two widely separated data rates (64 and 1024 KBPS). Typical simplified models of wire are broken into two regions with a boundary at about 200 kHz. Since the DVM operates with frequencies above and below this boundary it was decided a more sophisticated model was needed to include all effects.

The model has several defining parameters. The gauge of the cable can be selected as well as the length. The cable gauge indirectly specifies a set of coefficients used to compute the frequency dependent lumped constant parameters (R,L,G,&C in Htp in figure 4.2.) [Bell77]

It was hoped this model would allow us to determine over what distance the DVM system would operate. Unfortunately, the hardware described in chapter 5, does not give performance similar enough to the simulated cable to derive conclusions necessary to answer the length question from the simulations alone.

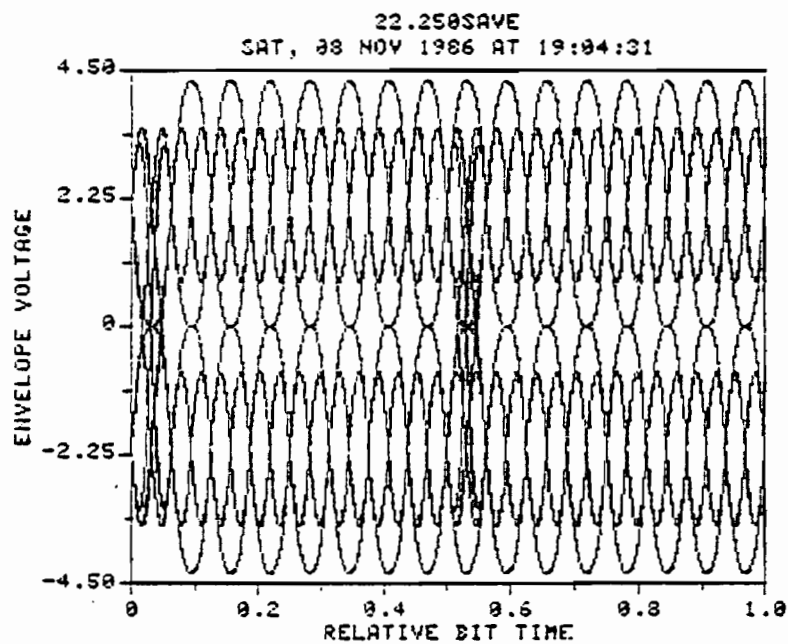


Figure 4-8. Eye diagram of 64 KBPS signal.

Figure 4.8 shows the eye diagram from a simulation using 250 feet 22 gauge of twisted pair. It shows why the eye pattern for the 64 KBPS data is difficult to use. The high rate data clutters it up. It appears that a schmitt trigger can recover the voice since there are no wild transitions through any portion of the eye. Any clock jitter will be dominated by data transitions occurring at the boundary of a voice bit transition. At most this amounts to one half a data bit which represents about 0.5 microseconds of jitter versus a period of 15 microseconds for a voice bit. This is hardly anything to worry about. Changing the voice data rate up to 128, 160 or 256 KBPS changes the ratio of these values and clock jitter becomes critical in the manchester decoding process. NOTE, that it does not matter one iota as far as de-multiplexing the two data streams is concerned. That process is analog in nature and independent of the clock information present.

The eye diagrams in figures 4.9 through 4.12 that follow cover two gauges of wire and two distances: 250 and 500 foot lengths of 22 and 24 gauge wire were simulated.

The interesting comparison to make is the difference between the 22 and 24 gauge pictures. The 22 gauge, as one would expect, gives better performance. The attenuation of the 24 gauge cable at higher frequencies is closing the eye of the repeating bit pattern considerably more in the 250 foot length. At 500 feet the 24 gauge eye is closed while the 22 gauge eye looks open enough to recover data.

4.4.3 Investigation of Clock Skew. The DVM system with the butterworth channel, is used to investigate the effect of creating a skew or phase

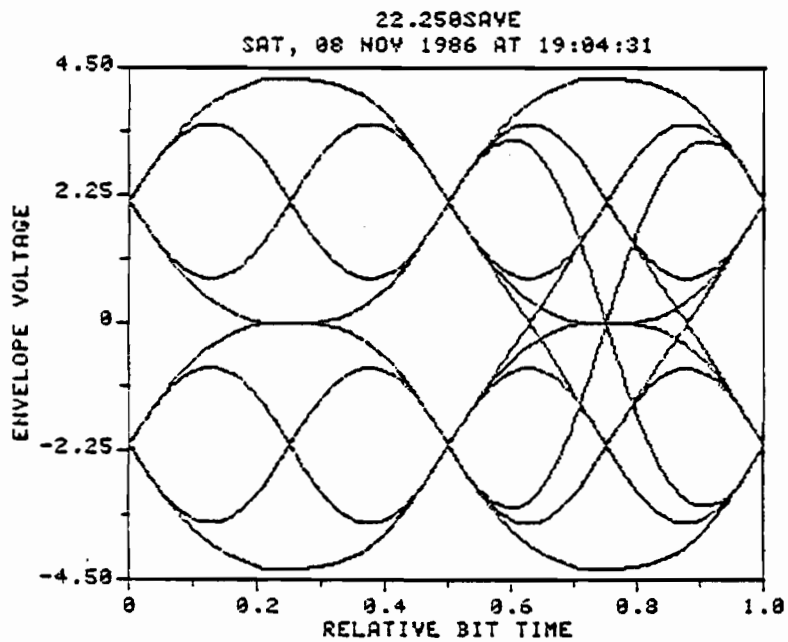


Figure 4-9. Eye diagram: 22 gauge, 250 ft cable.

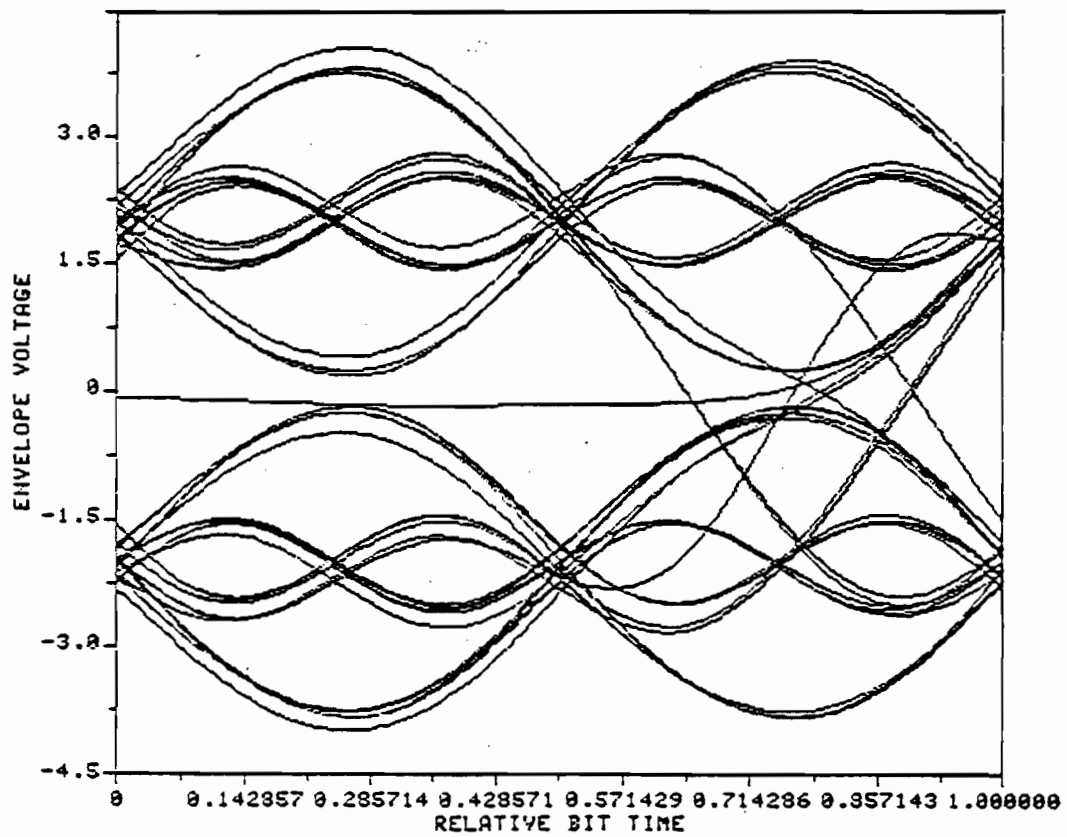


Figure 4-10. Eye diagram: 24 gauge, 250 ft cable.

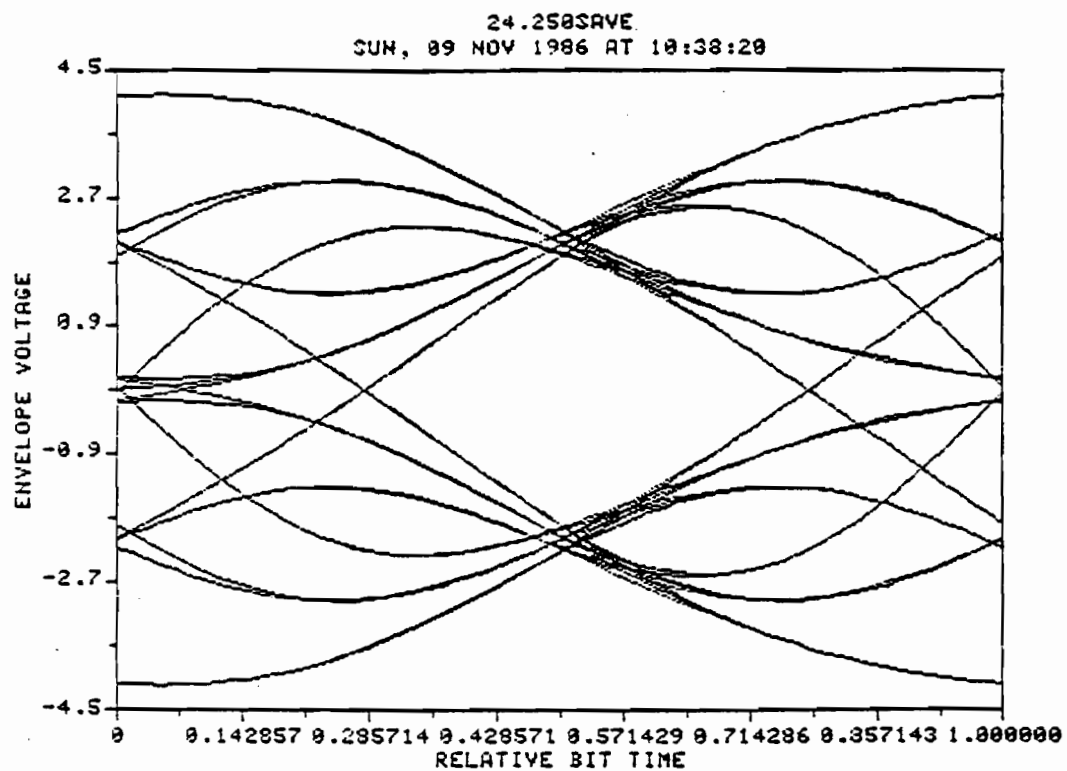


Figure 4-11. Eye diagram: 22 gauge, 500 ft cable.

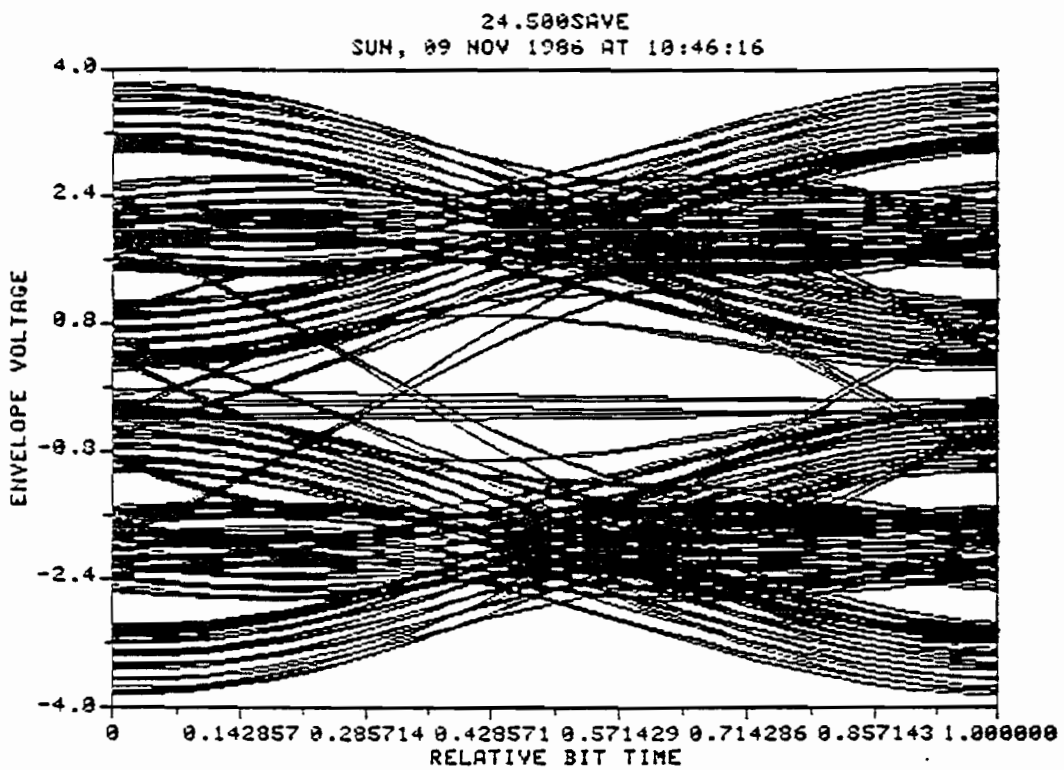


Figure 4-12. Eye diagram: 24 gauge, 500 ft cable.

offset between the high rate and low rate source clocks. It is important to know whether separate clocks can be used to drive the sources or whether a common clock is required.

Table 4.1 shows a table of clock skews versus errors for both voice and data channels. The voice clock was lagged by multiples of 5% from 10% to 75% of the data bit length. Thirty two bits of voice and 512 bits of data were simulated.

Skew %	Skew (ns)	Errors	
		Voice	Data
5	49	0	1
10	98	2	3
15	146	0	1
20	195	0	0
25	244	0	21
30	293	2	16
35	342	0	1
40	391	0	1
45	439	1	0
50	488	0	0
55	537	0	0
60	586	3	0
65	635	3	0
70	684	2	3
75	732	0	1

TABLE 4-1. Errors vs voice to data clock skew.

Examining the table we see the expected result that errors begin to occur when the skew approaches the 25% mark where the voice transitions interfere with the data at the data sampling point. † The number of

† Entries with three or fewer errors can be ignored because these are a result of the manchester decoder trying to get synchronized to the center cell transition. The simulation program prints the time location of the first ten errors and these were found to be in the first few initial bits.

errors quickly drops down again to zero following the sampling point.

Further simulations at lower channel bandwidths will probably show that the lower percents of skew will affect the performance more. In addition, errors will probably creep in around the 50% mark as the voice transitions add jitter to the data clock recovery.

4.5 Simulation Conclusions

The simulations showed that the DVM was worth building. They also provided some insight into the performance of the DVM under various channel bandwidth and clock skew limitations.

The eye diagrams show the system should work with channel bandwidths down to 1.0 to 1.5 MHz. More work needs to be done to get realistic model for the wire but the simulations show significant degradations occur as the cable length increases or if the cable gauge is increased. Clock skew does not seem to be problem since nearly 250 nanoseconds of skew is required to degrade system performance. Still, a common clock for driving the sources is indicated since two seperated clocks could easily start at random with 25% skews.

Next, the hardware prototype DVM is described along with some performance evaluations.

5. Data and Voice Multiplexer Prototype Hardware

This chapter describes the design, principles of operations (POP), and the performance of the DVM hardware prototype. The first half covers the design and the second half covers the performance.

5.1 POP - Principles of Operation

This section describes the hardware translation of the DVM concept. The sub-sections that follow treat each sub-system by describing how it was implemented in the prototype. An occasional diversion will describe differences between the hardware and simulation models. The six sub-systems covered are: sources, multiplexer, channel, de-multiplexer, manchester decoders, and error detection.

The prototype circuits are drawn on eight sheets located in appendix B. References to them are made by sheet number. (For example the signal DATA.CLK is found on sheet 1.) Component and signal location tables are found in Appendix C and D respectively.

5.1.1 Source Bit Streams. The sources consume the largest number of parts and board space and yet they are not even a part of the DVM proper! Three and a half pages of the schematic (sheets 1->4) contain the clock generation, voice and data PN generators, and manchester encoder circuits. Lets begin with the clock generation.

5.1.1.1 Clock generation Sheet 1 shows the clock generation schematic. A 20.48 MHz, ComClock TTL crystal clock, is the master system clock (CLK.20480). Two divider chains, made of binary and decade counters

connected to form synchronous counters, divide CLK.20480 to form the five different bit rate clocks required to test the DVM system.

The first chain consisting of U5 and U4, 741s163 synchronous binary counters, divides CLK.20480 by 128 to get a 160 kHz clock (CLK.160). The 160 kHz clock is the rate used for AT&T's DCP protocol.

The second chain consists of U1 and U2, 741s163's, and U3, a 741s162 synchronous decade counter. U3 provides an enable signal to U1 and U2 every ten clock pulses, thus dividing CLK.20480 by ten. U1 and U2 provide the additional divisions necessary to get the 256 kHz, 128 kHz, and 64 kHz clocks (CLK.256, CLK.128, and CLK.64 respectively) used for voice bit rate DVM performance evaluation. Each clock is connected to HD1 allowing it to be switched to the voice source PN generator.

The second divider chain also provides the 1024 kHz clock (DATA.CLK) used to drive the data source PN generator. The signals DATA.CLK and VOICE.CLK clock the shift registers of the PN generators on sheets 2 and 3 which are described next.

5.1.1.2 Pseudo Noise (PN) Bit Stream Generators. The simulations use software, uniform, pseudo random number generators to produce bit streams for testing the DVM. The hardware uses pseudo noise bit streams. These are generated using n bit shift registers with several taps combined and fed back to the shift register input [Smith85]. A pseudorandom sequence of $2^n - 1$ bits which has one more 1 bit than zero bits can be generated this way. A sequence of all zeroes is not allowed since this would freeze the generator and a continuous stream of zeroes would be output.

Long streams of zeros or ones are called runs. One-half the runs are of length one, one-fourth of length two, one-third of length eight etc. Two PN sources are required: one for the voice stream and one for the data stream. Their design is identical and shown on schematic sheets 2 and 3.

U7 and U8 (74ls374 8 bit registers) are connected to form a sixteen bit shift register. Only fourteen bits of the shift register are used in the PN generator, leaving two stages left to provide delayed versions. Feedback is from taps at 2, 12, 13 and 14 bits. The taps are combined in U6 (74ls86 exclusive or) and fed back to the first bit of the shift register.

A method is needed which protects against the possibility of the shift register powering up filled with zeroes. U9 (74ls163) is used to detect when a sequence of all zeroes is about to occur. The clock driving the shift register is also driving U9 to count the number of 0's appearing in the output stream. It does this by allowing any 1's in the stream to force the counter to load a value of 0001 and continue counting. When fourteen 0's occur the count will reach fifteen causing the ripple carry to go high. This causes the output-enable on the shift register (U7, U8) to be disabled making the outputs of U7 and U8 go to a high impedance state. A pull-up resistor on the second to last tap forces a 1 to appear at the last shift register stage. At the next clock edge this gets fed back to the front of the shift register and the sequence will start up.

DPDT switch S1 allows either constant or PN bits to appear at the inputs to the manchester encoders (TX.VOICE and TX.DATA). The outputs

of the PN generators (PN.VOICE and PN.DATA) are connected to S1 (sheet 4) along with constant voltages: Plus five volts for the data signal and zero volts for the voice signal. The constant voltages will be manchester encoded into a continuous stream of 1's (data) or 0's (voice).

5.1.1.3 Manchester encoders. Manchester encoding is done for a source by combining the source clock with the source NRZ bit stream using an exclusive-or just like the simulation model. U6 (741s86 XOR) combines VOICE.CLK and TX.VOICE to form INT.MCV (internal manchester coded voice). U11 combines DATA.CLK and TX.DATA to form INT.MCD. Switch S2 selects either the internal manchester coded streams (INT.MCV or INT.MCD), or external manchester coded streams (EXT.MCV or EXT.MCD) from front panel DB-25 connector J1. The switched signals are called MCV and MCD and are routed to the input of the multiplexer described next.

5.1.2 Multiplexer. The multiplexer is certainly the simplest portion of the entire DVM prototype since it consists of only one IC. Its operation is simulated the same way it is implemented. The two manchester encoded sources are coded to drive the PLUS and MINUS signals transmitted over the twisted pair.

The multiplexer, U15 (741s153 dual four to 1 multiplexer), is wired as two four selection coders. The multiplexers (Mux1 and Mux2) have common select inputs (B and A) driven by the signals MCV and MCD respectively. The two bit combination of MCV and MCD allows selection from one of the four inputs on Mux1 or Mux2. The four inputs are wired according to the following table.

MCV (B)	MCD (A)	Mux1 (PLUS)	Mux2 (MINUS)
0	0	0	0
0	1	0	1
1	0	0	0
1	1	1	0

TABLE 5-1. U15 multiplexer input coding

The output of Mux1 drives the PLUS signal and Mux2 drives the MINUS signal. These signals are passed on to the line drivers discussed next in the channel circuitry section.

5.1.3 Channel Circuitry (Line Drivers). There are four major "components" to the channel section. These are line drivers, transmit side transformer, twisted pair, and receive side transformer. The channel components have proved the most difficult to model in the simulations. They also present certain problems in implementation. For instance the twisted pair must be terminated in its characteristic impedance at the receive side and at the transmit side line drivers.

The line drivers (sheet 4) are little more than electronic single pole double switches implemented using a couple of transistors. (Note the PLUS and MINUS drivers are identical so only one will be described.) U17 and U10 are used to provide true and complemented values of the PLUS signal. These signals drive the totem pole arrangement of Q3 and Q4 forming the SPDT switch. The switch connects the 50 ohm impedance matching resistor R3 to either VCC (+5) or GND. The other terminal of R3 connects to the DOT terminal on pulse transformer T1. Transformer operation is possible in this circuit because the three level DVM signal

has no DC component in its spectrum (see chapter 2.)

An identical circuit is connected to the MINUS signal from the multiplexer. The function of these two circuits is to force current in either direction through T1. This action develops a signal across the receive side transformer impedance matching resistor R4 with a swing of five volts.

The twisted pair is connected via RJ11, 4 pin modular telephone jacks J2 and J3. The jacks connect to the pulse transformers at both ends of the system.

The output of receive transformer T2 is connected to the input of a differential amplifier (gain = 2) formed from R5-R8, C1, C3 and U16 (LM318 - high slew rate operational amplifier {opamp}) which provides the difference (RCVSIG) of the PLUS and MINUS signals. RCVSIG is passed on to the de-multiplexer section which recovers the voice and data manchester coded signals.

5.1.4 De-Multiplexer. The de-multiplexer system, like its simulation counterpart, consists of an absolute value circuit and a schmitt trigger. The absolute value circuit recovers the manchester encoded data signal (R.MCD) from the three level signal RCVSIG. The schmitt trigger recovers the voice signal (R.MCV). The implementation of the schmitt trigger (sheet 5) is described in [Metz].

The absolute value circuit consists of a full wave bridge (D1-D4) followed by a difference amplifier (R9-R12, C2, C4 and U18) The difference amplifier is required because the input signal is ground referenced so the output of the bridge cannot be ground referenced. It

might be possible however to take the floating output of T1 and connect it directly to the bridge. This would eliminate the seven components in the diff-amp. The output of opamp U18 drives a 74ls14 schmitt trigger which takes the analog output of the absolute value circuit and regenerates a binary signal from it. (This schmitt trigger is modeled in the simulation)

Manchester decoding of R.MCD and R.MCV is performed using several IC's and discrete components as discussed next.

5.1.5 Manchester Decoders. The manchester decoders are implemented in the same manor as the simulation models. Operation is probably not quite as "ideal" because two parts of the decoders, the delay and the timer operation, are made partially using analog components which are difficult to model. Nearly identical circuits are used to recover the NRZ voice and data streams. The only difference being the resistor values used to define the timer on time.

Two schmitt trigger inverters from U19 provide approximately thirty to forty nano-seconds of propagation delay to the incoming manchester encoded to form MCD.DLY (refer to figure 3.6 pg 36 for signals). MCD.DLY is XOR'ed by U21 with R.MCD to generated narrow pulses at each rising and falling edge of R.MCD. These pulses are used to trigger the timer made with the non-retriggerable configured monostable multivibrator U22. The non-retriggerable mode causes U22 to filter out every other trigger pulse, keeping the center transitions from the manchester encoded waveform. (Note: this requires that a 1->0 or 0->1 bit sequence is transmitted so the timer can become synchronized to the

center transitions.) Every time it is triggered, the Q bar output of the mono-stable goes low for 75% of a bit time.

R22 and C5 determine the timer on length. The TTL data book gives curves and formulas for determining these values but I found it easier to select the capacitor to get the time in the right range then use a multi-turn potentiometer (R22 for data decoder, R28-R29 for voice) to trim the value to 75% on time with the circuit running.

The rising edge of the Q bar output (D.CLK) clocks a D-flipflop (U23) causing it to sample D.MCD one quarter of the way into a bit. Recall that a manchester encoded 0 bit is sent as a 10 and a 1 bit is sent as a 01. Consequently, if the flipflop samples a 1 then the received bit is a 0 bit, otherwise it should be a 1 bit. This is why the output bit stream is taken from the Q bar output of the flipflop.

The voice decoder differs from the data decoder in this respect since it has an additional stage of inversion after the schmitt trigger in the demultiplexer. The recovered NRZ bit stream for the voice system is take from the Q output of U23.

The last remaining system is the error detection subsystem which takes the recovered NRZ bit streams DATA.BITS and VOICE.BITS and compares them to the originally transmitted streams TX.DATA and TX.VOICE.

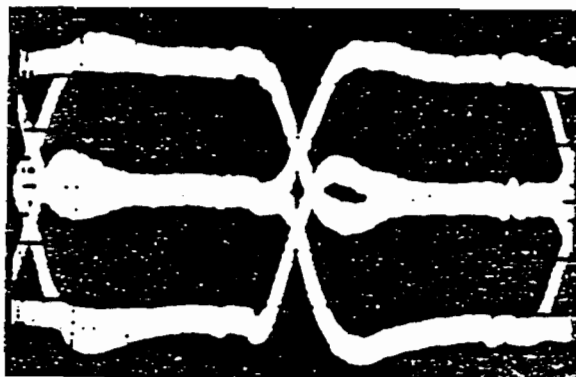
5.1.6 Error Detection System. The error detection system is implemented in a manor similar to the simulation code. The transmitted bit stream is compared continuously with the recovered bit stream using an exclusive-or gate. The exclusive-or output is sampled at the falling

edge of the respective bit stream transmit clock with a D-flipflop. When the streams differ, a 1 is clocked into the flip flop causing the error signal (D.ERR or V.ERR) to go high. D.ERR or V.ERR can be connected to frequency counter to find the rate at which errors are occurring. Note that if the channel and decoding delays exceed one half a bit period then this system will not work because the transmit and recovered streams will not overlap with the proper values at the sample time. Fortunately, the delay was just under the half bit period in the high speed data channel.

5.2 Performance

The prototype circuitry was constructed in a Tektronix TM-500 series blank plug-in unit. The whole unit is inserted into a TM-500 instrument mainframe which supplies power to power supplies on the plug-in. Operation begins when the TM-500 power is turned on and a cable is connected to J1 and J2 (RJ11 phone jacks). S1 is set to random or constant data and S2 is set to internal manchester encoding. Last, a bit rate for the voice channel is selected by rotating switch S3 to 64 KBPS, 160 KBPS, or 128 KBPS.

Changing the length of the cable affects the operation of both data and voice channels. Selecting a higher voice bit rate also affects the error rate. (Performance of the 160 KBPS system is discussed separately in the next chapter.) A secondary, more subjective measure, is eye diagrams obtained by photographing oscilloscope displays of the output signals. Figure 5.1 shows the eye diagram obtained by running the DVM with 100ft of parallel conductor phone cable.



Frequency response
of the hardware 100ft ch.

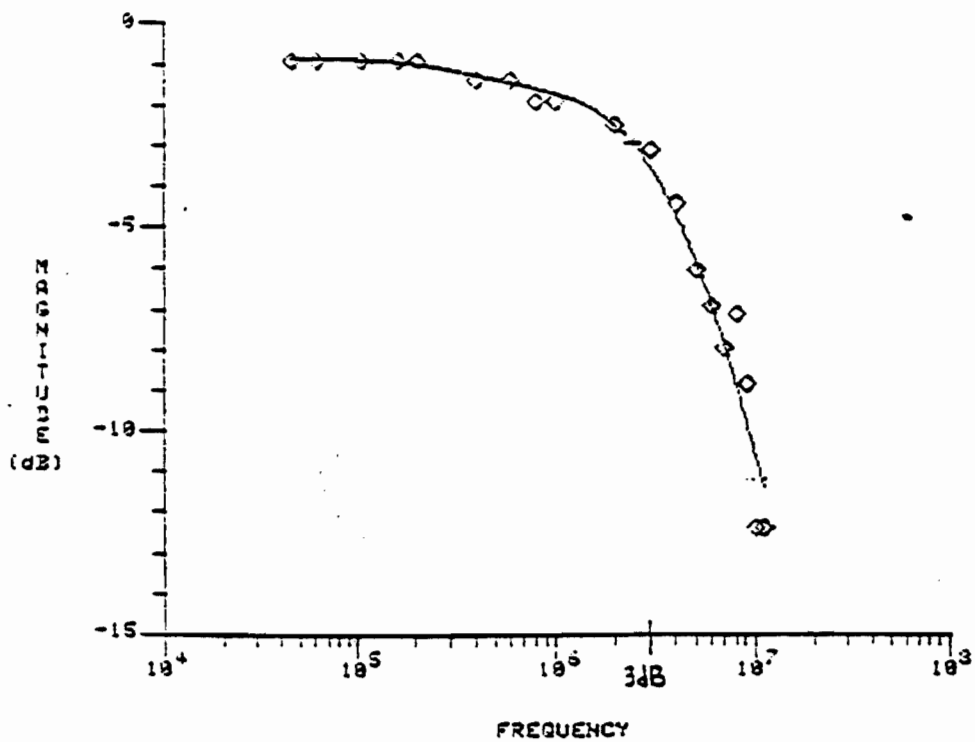


Figure 5-1. Hardware performance with 100ft of cable.

In a nutshell, the results are the DVM prototype works over 100ft of parallel conductor phone wire at the 64 and 128 KBPS voice rates plus 1024 KBPS data rate with zero errors. The 1024 KBPS channel showed no errors over a 24 hour operation period when operating with the 64 KBPS voice rate over the same 100ft cable. Figure 5.1 shows the eye diagram of RCVSIG and the transfer characteristic of the channel (Transformers + cable).

Changing to a longer stretch of 250ft 24 gauge four conductor shielded cable gave the same results. The DVM would not work with a length of 750ft. The bandwidth of the cable probably decreased below the 2.75 MHz so the manchester decoders were not getting enough signal to work properly.

Next the performance of simulations and hardware using the DCP voice rate are examined.

6. Operation and Performance at DCP Rate.

6.1 The Vee Problem.

Operating the DVM at a voice rate of 160 KBPS (DCP rate) causes problems since the voice and data signals are no longer orthogonal. To be orthogonal, the data rate must be an integer multiple of the voice rate so the signal edges line up. As a result, when the low speed (160 KBPS) signal makes a transition it will occur occasionally in the inner portion of a data bit. When the data signal is demultiplexed by the absolute value, the voice transition in the data makes a notch or "vee" in the data. If the vee is too wide and occurs next to a sample point, the manchester decoder will incorrectly decode the sampled value.

Figure 6.1 shows oscilloscope pictures of the problem. The top picture shows constant manchester coded signals and the bottom shows PN manchester data. The bottom picture would normally have many more vees in it but it was taken with the scope time base slowed and the 10x expansion on. This allowed me to scan over the magnified eye diagram and pick out a vee to illustrate the problem.

The demultiplexing process is not affected by the vee problem. Only the recovery of the NRZ data from the demultiplexed manchester encoded data is. The problem caused by the vee can be minimized if it is made narrower or if the thresholds of the schmitt trigger following the absolute value are lowered. Unfortunately, as the channel bandwidth decreases (cable gets longer) the slew rate of the signals decreases and this tends to widen the vee. If the thresholds are lowered then noise

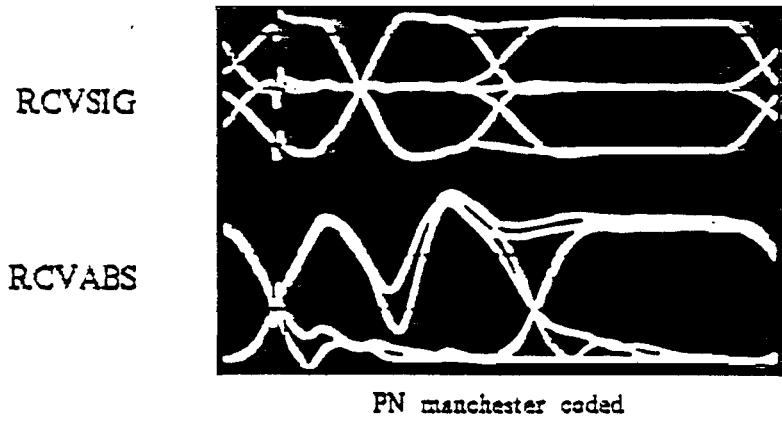


Figure 6-1. DCP rate 'vee' problem.

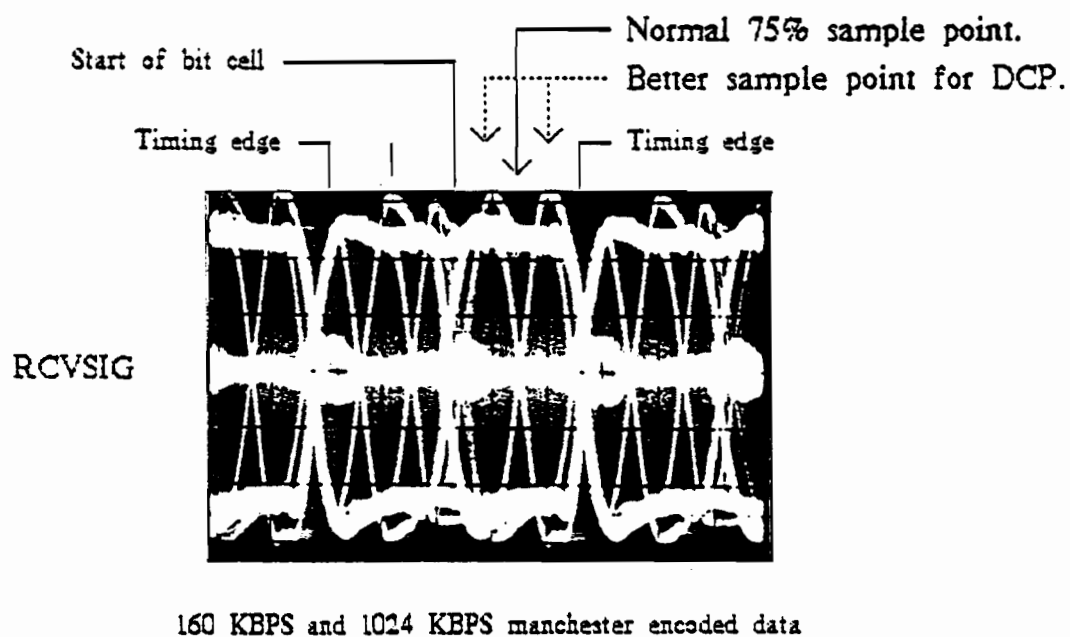


Figure 6-2. Illustration of sample point location for DCP.

A. SYSTID SIMULATION CODE

A.1 DAVIS06

```

+++++
+
+          DAVIS06          +
+
+++++

```

SYSTEM: Data and Voice Integration System (DAVIS06)

Signals

clkv - 50% duty cycle square wave clock for voice circuit.
 clkd - 50% duty cycle square wave clock for data circuit.

txd - NRZ transmitted data.
 txv - NRZ transmitted voice.
 mcv - Manchester encoded voice.
 mcd - Manchester encoded data.

minus - Minus line multiplexer output
 plus - Plus line multiplexer output

rcv - Trinary waveform received over wires

vmc - Reconstructed manchester voice signal (in rcvr)

rcvabs- Absolute value of rcv signal.
 dmc- Reconstructed manchester data signal (in rcvr)

vmcdly- Vmc signal delayed a few DT's
 dmcdly- Dmc signal delayed a few DT's

vtrig - Gives a pulse for every rising/falling edge of vmc
 dtrig - Gives a pulse for every rising/falling edge of dmc

vclk - Sample clock used to recover NRZ voice signal
 dclk - Sample clock used to recover NRZ data signal
 vrcv - NRZ voice signal
 drcv - NRZ data signal

REAL: clkv,txv,clkd,txd,mcv,mcd,minus,plus,rcv,clk,
 \$ vmc,vmcdly,vtrig,vclk,vckbar,vrcv,
 \$ rcvabs,dmc,dmcdly,dtrig,dclk,dckbar,drcv,
 \$ highsg,lowsig,lvclk,ldclk,cmpv,cmpd,chanin
 VSIZE: 32000

```

REAL tbv,tbd,vrate,drate,rt,zeropi,vltl,vhtl,
$   tmrpct, chcf, chbw, chgain, slewrt, high, low, ref, dltl,
$   dhtl, dtmr, vtmr, drlvl, ve_time(10), de_time(10),
$   start, stop, simbw, vang
INTEGER nstv, nstd, nsh, nbits, dly, order, lowpas, vtimer, deltaP,
$   dtimer, derr, verr, chordr, is1, is2, isv, isd, i, j, lagpct, lag
CHARACTER*40, pname, simnam
DEFAULT: lowpas=1, zeropi=0.0

```

CALCULATE:

```

read *,pname,simnam
if (simnam.ne.'DAVIS06') then
  print *, '## Incorrect simulation name in data file.'
  stop
end if

read *,pname,simbw
read *,pname,start,stop
read *,pname,high,low,ref
read *,pname,vltl,vhtl
read *,pname,dltl,dhtl
read *,pname,drlvl
read *,pname,vrate,drate
read *,pname,vtmr,dtmr
read *,pname,isv
read *,pname,isd
read *,pname,chbw
read *,pname,chordr
read *,pname,chgain
read *,pname,deltaP

vrate = vrate * 1000.0
drate = drate * 1000.0

tbv = 1.0/vrate
tbd = 1.0/drate

dt = 1.0/simbw

tstart = start/1.0e6
tstop = stop/1.0e6

vtmr = vtmr / 100.0
dtmr = dtmr / 100.0
vtimer = nint(vtmr*tbv/dt)
dtimer = nint(dtmr*tbd/dt)

dly = 5

is1 = isv

```

```

is2 = isd
vang = 0.
chbw = chbw * 1.0e6
rt = 1.0/(2.0*chbw)
slewrt = (high-low)/rt
chcf = 0.0

```

```

| Place all SYSTID "VARY" loops before the error counters.
| They must be reset before every run.

```

```
VARY: lagpct = 5,75,deltaP
```

```

lag = int((lagpct)/100.*tbd/dt)
print *,'lagpct',lagpct,'lag',lag
isv = is1
isd = is2
derr = 0
verr = 0
do i=1,10
    ve_time(i) = 0.0
    de_time(i) = 0.0
end do

```

```

| Main system model begins here.

```

```
SIMULATE:
```

```

highsg = high
lowsig = low

```

```

| Manchester encoded voice source.

```

```

null > sq(vrate,vang) > clk
clk > delay(lag) > clkv
clkv = (clkv*0.5+0.5) * high
clkv > ranpls(high,low,ref,is1) > txv
clkv,txv > xor(high,low,ref) > mcv

```

```

| Manchester encoded LAN data source.

```

```

null > sq(drate,zeropi) > clkd
clkd = (clkd*0.5+0.5) * high
clkd > ranpls(high,low,ref,is2) > txd
clkd,txd > xor(high,low,ref) > mcd

```

```

| Multiplexer.

```

```

mcv,mcd,lowsig,highsg,lowsig,lowsig >
$ mplexr(drvlvl,low,ref) > minus,null

```

```

    mcv,mcd,lowsig,lowsig,lowsig,highsg >
$    mplexr(drvlvl,low,ref) > plus,null

receiver
    chanin = (plus - minus)*0.48

Channel model
    chanin > butwth(chordr,lowpas,chcf,chbw,chgain) > rcv
    rcv = rcv * 2.0

Demultiplex the voice channel.
    rcv > schmitt trigger(high,low,ref,vltl,vhtl) > vmc

Perform manchester decode on recovered voice stream.
    vmc > delay(dly) > vmcdly
    vmc,vmcdly > xor(high,low,ref) > vtrig
    vtrig > timer(high,low,ref,vtimer) > vckbar,vclk
    highsg,highsg,vclk,vmc >d flip flop(high,low,ref)> vrcv,null

Voice channel error measurements.
Compare transmitted and received
NRZ streams on the falling edge of the tx clock.

    txv,vrcv > xor(high,low,ref) > cmpv
    if ((lvclk.gt.ref).and.(clkv.le.ref).and.(cmpv.gt.ref)) then
        verr=verr+1
        if ((verr.le.10).and.(verr.gt.0)) ve_time(verr)=time

    end if

    lvclk = clkv

Demultiplex the LAN data.
    rcvabs = abs(rcv)
    rcvabs > schmitt trigger(high,low,ref,dltl,dhtl) > dmc

Perform manchester decoding on recovered LAN data.
    dmc > delay(dly) > dmcdly
    dmc,dmcdly > xor(high,low,ref) > dtrig
    dtrig > timer(high,low,ref,dtimer) > dckbar,dclk
    highsg,highsg,dclk,dmc >d flip flop(high,low,ref)> drcv,null

Data channel error measurements.
Compare transmitted and received
NRZ bit streams on the falling edge of the transmit clock.

```

```

write(i$out,3) vtmr*100.0,dtmr*100.0
3 format(2x,
$      '### Manchester decoder timers (% of bit time)',/,
$      2x,'      Voice = ',f4.1,'% , Data ',f4.1,'% ',/)

write(i$out,4) slewrt/1.0e6,rt/1.0e-9,chordr,chbw/1.0e6,chgain
4 format(2x,'### Channel Model parameters',/,
$      2x,'      Slew rate = ',f5.1,' (volts/micro-sec)',/,
$      2x,'      Rise time = ',f5.1,' (ns)',/,
$      2x,'      Filter order = ',i1,/,
$      2x,'      Bandwidth = ',f5.2,' (Mhz)',/,
$      2x,'      Channel gain = ',f5.3,/)

write(i$out,5) dltl,dhtl
5 format(2x,'### Schmitt trigger levels in data circuit.',/,
$      2x,'      Low level = ',f5.2,' (volts)',/,
$      2x,'      High level = ',f5.2,' (volts)',/)

write(i$out,6) vltl,vhtl
6 format(2x,'### Schmitt trigger levels in voice circuit.',/,
$      2x,'      Low level = ',f5.2,' (volts)',/,
$      2x,'      High level = ',f5.2,' (volts)',/)

write(i$out,7) high,low,ref
7 format(2x,'### Logic levels for signals ',/,
$      2x,'      High level outputs = ',f5.2,/,
$      2x,'      Low level outputs = ',f5.2,/,
$      2x,'      Reference level = ',f5.2,/)

write(i$out,9) drvlvl
9 format(2x,'### Line driver high output voltage = ',f3.1,
$      ' (volts)',/)

write(i$out,10) isv,isd
10 format(2x,'### Random number seeds: Voice = ',i15,
$      ' Data = ',i15,/)

write(i$out,11) lagpct,float(lagpct)/100.*tbd*1.0e9,vang
11 format(2x,'### Voice to Data clock lag = ',I3,
$      ' % of Data bit time',/,
$      2x,'      = ',f7.2,' (ns)',/,
$      2x,'      = ',f6.2,' (degrees)',/)

END: davis

```

A.2 DAVIS07

```

+++++
+                                     +
+                               DAVIS07 +
+                                     +
+++++
SYSTEM: Data and Voice Integration System (DAVIS07)

```

Signals

clkv - 50% duty cycle square wave clock for voice circuit.
 clkd - 50% duty cycle square wave clock for data circuit.

txd - NRZ transmitted data.
 txv - NRZ transmitted voice.
 mcv - Manchester encoded voice.
 mcd - Manchester encoded data.

minus - Minus line multiplexor output
 plus - Plus line multiplexor output

chanin - plus - minus signals. Input to transmission line
 rcv - Channel output.

vmc - Reconstructed manchester voice signal (in rcvr)

rcvabs- Absolute value of rcv signal.
 dmc - Reconstructed manchester data signal (in rcvr)

vmcdly - Vmc signal delayed a few DT's
 dmcdly - Dmc signal delayed a few DT's

vtrig - Gives a pulse for every rising/falling edge of vmc
 dtrig - Gives a pulse for every rising/falling edge of dmc

vclk - Sample clock used to recover NRZ voice signal
 dclk - Sample clock used to recover NRZ data signal
 vrcv - NRZ voice signal
 drcv - NRZ data signal

```

REAL: clkv,txv,clkd,txd,mcv,mcd,minus,plus,rcv,txddly,txvdly,
$   vmc,vmcdly,vtrig,vclk,vckbar,vrcv,diff,trfout,cblout,
$   rcvabs,dmc,dmcdly,dtrig,dclk,dckbar,drcv,
$   highsg,lowsig,lvclk,ldclk,cmpv,cmpd,chanin
SAVE [davis07] rcv
VSIZE: 32000
REAL tbv,tbd,vrate,drate,rt,zeropi,vltl,vhtl,length,
$   tmrpt,trnfcf,trnfbw,gain,slewrt,high,low,ref,ltl,

```

```

$   htl,dtmr,vtmr,drv1v1,ve_time(10),de_time(10),
$   start,stop,simbw,vang,flotx,fhitx,florx,fhirx
INTEGER nstv,nstd,nsh,nbits,dly,order,lowpas,vtimer,hipass,
$   numpts,gauge,lag,
$   dtimer,derr,verr,chordr,is1,is2,i,j,
$   gaug19,gaug22,gaug24,gaug26
DEFAULT: gaug19=1,gaug22=2,gaug24=3,gaug26=4,
$   lowpas=1,hipass=2,zeropi=0.0
CHARACTER*18 filnam,rdwrit
CHARACTER*40 pname,simnam
CHARACTER*8 cable(4)
Data (Cable(i),i=1,4)
$   /'19 gauge','22 gauge','24 gauge','26 guage'/

()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
CALCULATE:

```

```

read *,pname,simnam
if (simnam.ne.'DAVIS07') then
  print *,'## Incorrect simulation name in data file.'
  stop
end if

```

```

read *,pname,simbw
read *,pname,high,low,ref
read *,pname,v1t1,vhtl
read *,pname,l1t1,htl
read *,pname,drv1v1
read *,pname,vrate,drate
read *,pname,start,stop
read *,pname,vtmr,dtmr
read *,pname,filnam
read *,pname,rdwrit
read *,pname,flotx,fhitx
read *,pname,florx,fhirx
read *,pname,gauge
read *,pname,length
read *,pname,is1,is2
read *,pname,lag

```

```

vrate = vrate * 1000.0
drate = drate * 1000.0
tbv = 1.0/vrate
tbd = 1.0/drate
vang = zeropi
dt = 1.0/simbw
tstart = start/1.0e6
tstop = stop/1.0e6
vtmr = vtmr / 100.0
dtmr = dtmr / 100.0
vtimer = nint(vtmr*tbv/dt)

```



```

dtimer = nint(dtmr*tbd/dt)
dly = 5
derr = 0
verr = 0
numpts = 4096

|
| BEGIN
SIMULATE:

    highsg = high
    lowsig = low

|
| Sources
| Voice
| null > sq(vrate,vang) > clkv
| clkv = (clkv*0.5+0.5) * high
| clkv > ranpls(high,low,ref,is1) > txv
| clkv,txv > xor(high,low,ref) > mcv

| LAN data
| null > sq(drate,zeropi) > clkd
| clkd = (clkd*0.5+0.5) * high
| clkd > ranpls(high,low,ref,is2) > txd
| clkd,txd > xor(high,low,ref) > mcd

|
| Multiplexer
| mcv,mcd,lowsig,highsg,lowsig,lowsig >
$      mplexr(dr1vl1,low,ref) > minus,null
| mcv,mcd,lowsig,lowsig,lowsig,highsg >
$      mplexr(dr1vl1,low,ref) > plus,null

|
| Channel modeling: Transformer -> twisted pair -> Transformer
| chanin = (plus-minus)*.48
| chanin >
$ chanl(rdwr1t,filnam,numpts,flotx,fhitx,florx,fhitx,gauge,length)
$      > rcv

| Receiver front end - (Differential amp gain in hardware)
| rcv = rcv * 2

|
| voice decoding
| rcv > schmitt trigger(high,low,ref,v1tl,vhtl) > vmc
| vmc > delay(dly) > vmcdly
| vmc,vmcdly > xor(high,low,ref) > vtrig
| vtrig > timer(high,low,ref,vtimer) > vckbar,vclk
| highsg,highsg,vclk,vmc > d flip flop (high,low,ref) > vrcv,null
|

```

```

| voice channel error measurements.
| Compare transmitted and received
| streams on the falling edge of the tx clock.
|
txv > delay(lag) > txvdly
txvdly, vrcv > xor(high, low, ref) > cmpv
if ((lvclk.gt.ref).and.(vclk.le.ref).and.(cmpv.gt.ref)) tl
    verr=verr+1
    if ((verr.le.10).and.(verr.gt.0)) ve_time(verr)=time
end if

lvclk = vclk

| data decoding
rcvabs = abs(rcv)
rcvabs > schmitt trigger(high, low, ref, ltl, htl) > dmc
dmc > delay(dly) > dmcdly
dmc, dmcdly > xor(high, low, ref) > dtrig
dtrig > timer(high, low, ref, dtimer) > dckbar, dclk
highsg, highsg, dclk, dmc > d flip flop (high, low, ref) > drcv, nu

| data channel error measurements.
| Compare transmitted and received
| streams on the falling edge of the transmit clock.
|
txd > delay(lag) > txddly
txddly, drcv > xor(high, low, ref) > cmpd
if ((ldclk.gt.ref).and.(dclk.le.ref).and.(cmpd.gt.ref)) then
    derr = derr+1
    if ((derr.gt.0).and.(derr.le.10)) de_time(derr)=time
end if

ldclk = dclk

```

CALCULATE:

```

write(i$out, 99) simnam
99 format(1x, /,
$ 2x, '### ', a40, /,
$ 2x, ' This system attempts to model the channel using a', /,
$ 2x, ' cable impulse reponse and convolving it with the', /,
$ 2x, ' input signal. ', /)

write(i$out, 98) verr, nint(vrate*tstop), derr, nint(drate*tstop)
98 format(2x, '### Error counts - Ignore if signal delays are not',
$ ' considered', /,
$ 2x, ' Voice errors: ', i4, ' in ', i4, ' bits', /,
$ 2x, ' Data errors: ', i4, ' in ', i4, ' bits', /)

```

```

if (verr.gt.0) then
  i=10
  if (verr.lt.10) i=verr
  write(i$out,97) (ve_time(j)*1.0e6,j=1,i)
  end if

97 format(2x,'### Voice error times of occurrence (micro-sec)',/,
$      2x,' ',10(f6.1,1x),/,/)

if (derr.gt.0) then
  i = 10
  if (derr.lt.10) i=derr
  write(i$out,96) (de_time(j)*1.0e6,j=1,i)
  end if

96 format(2x,'### Data error times of occurrence (micro-sec)',/,
$      2x,' ',10(f6.1,1x),/,/)

write(i$out,95) 1.0/(dt*1.0e6)
95 format(2x,'### Simulation bandwidth = ',f7.3,' (Mhz)',/)

write(i$out,1) vrate/1000.0,drate/1000.0
1 format(2x,'### Source rates',/,
$      2x,' Voice (kpbs)',f6.1,/,
$      2x,' Data (kpbs) ',f6.1,/)

write(i$out,3) vtmr*100.0,dtmr*100.0
3 format(2x,'### Manchester decoder timers (% of bit time )',/,
$      2x,' Voice = ',f4.1,'% ', Data ',f4.1,'% ',/)

write(i$out,4) flotx,fhitx,florx,fhitx,length,cable(gauge)
4 format(2x,'### Channel Model parameters',/,
$      2x,' Transformer bandpass filter cutoff frequencies',/,
$      2x,' Transmit = ',e10.3,', ',e10.3,/,
$      2x,' Receive = ',e10.3,', ',e10.3,/,
$      2x,' Cable length = ',f5.1,' (feet)',/,
$      2x,' Cable wire is ',a,/)

write(i$out,5) ltl,htl
5 format(2x,'### Schmitt trigger levels in data circuit.',/,
$      2x,' Low level = ',f5.2,' (volts)',/,
$      2x,' High level = ',f5.2,' (volts)',/)

write(i$out,6) vltl,vhtl
6 format(2x,'### Schmitt trigger levels in voice circuit.',/,
$      2x,' Low level = ',f5.2,' (volts)',/,
$      2x,' High level = ',f5.2,' (volts)',/)

write(i$out,7) high,low,ref
7 format(2x,'### Logic levels for signals ',/,
$      2x,' High level outputs = ',f5.2,/,

```

```
$          2x,'   Low level outputs = ',f5.2,/,  
$          2x,'   Reference level   = ',f5.2,/)
```

```
write(i$out,9) drlvl  
9  format(2x,  
$   '### Line driver high output voltage = ',f3.1,' (volts)',/)
```

```
write(i$out,10) is1,is2  
10 format(2x,'### RANPLS seeds: IS1 = ',i11,', IS2 = ',i11,/) )
```

```
END: Data and voice integration system
```

A.3 TWISTED PAIR CHANNEL

```

*=====
* NAME: CHANL                      AUTHOR: Tim Davis DATE: Jun,86
*-----
* PURPOSE: This model simulates a channel consisting of two
* transformers and a twisted pair transmission line.
* The signal enters a transformer, travels through the
* transmission line and finally leaves through another transformer.
* The transformers are modeled as bandpass filters while
* the transmission line is determined by an exponentially based
* transfer characteristic. The primary twisted pair constants
* R,L,G,C and Length are the parameters used to specify
* the transmission line properties. Note the general case
* where R,L,G and C vary with frequency is taken into account.
* The final baseband transfer characteristic is obtained
* by multiplying all the transfer characteristics together.
* Next, the inverse FFT is taken to get the channel's
* impulse response. The channel output is then found by
* convolving the impulse response with the channel input signal.
* The convolution is implemented using the tapped delay
* line model NEWTAP. The impulse response is saved in a
* file for future usage.
* The transfer function is calculated from  $-1/(2*dt)$  to  $1/(2*dt)$ .
* To perform the discrete convolution, the impulse response must be
* time shifted to the center of the window, resulting in an
* arbitrary time delay in the output signal.
*-----
*
*                      PARAMETER DEFINITION
* NAME - DESCRIPTION | TYPE | CLASS | RANGE
*-----
* rdwrit - read/write h(t) from/to disk file | cha*18 | input | 'READ' or
*                                             |       |       | 'WRITE'
* filnam - filename to read/write h(t) to. | cha*18 | input |
* (This file contains the impulse
* response)
* numpts - number of impulse resp. samples | integer | input | even<5000
* Also the number of transfer function
* samples. (must be a composite # with
* max prime factor = 127)
* FlowTX,PhiTX - Bandpass cutoff freq for | real*4 | input | >0<1.0/dt
* the transmit side transformer |
* <1.0/(2*dt)
* FlowRX,PhiRX - Bandpass cutoff freq for | real*4 | input | >0<1.0/dt
* the receive side transformer |
* <1.0/(2*dt)
* Gauge - Enumerated value specifying the | integer | input | 1,2,3,4
* cable gauge. 1=19gauge, 2=22gauge
* 3 = 24gauge, 4 = 26gauge
* Length - Length of the transmission line | real*4 | input | >0 feet
*-----

```

```

*           subroutines required
*   name           |           description
* -----|-----
*   getmax         |
*   fft            |
*   newtap         |
*   arshft        |
* -----|-----

```

```

MODEL:TX > CHANL(rdwrit,filnam,numpts,flotx,fhitx,florx,fhirx,gauge,
$           length) > RCV

```

```

integer      maxh
parameter    (maxh=5000)
complex:     win, wout
complex      h,gamma,txline,bandpass
real         flotx,fhitx,florx,fhirx,length,omega,freq,hdb,
$           realh(maxh),imgh(maxh),impres(maxh),delfrq,hmag
logical      op,fstm
integer      yes,no,ind,dep,gauge,gauge19,gauge22,gauge24,gauge26,
$           Funit,numpts,nhalf,i,mxindex,nshift
parameter    (yes=0,no=1,gauge19=1,gauge22=2,gauge24=3,gauge26=4)
character*18 filnam,rdwrit
stack<i>     init

```

```

calculate: *****

```

```

if(init.eq.Yes)then
  print *, 'DEBUG: rdwrit = ',rdwrit
  print *, 'DEBUG: filnam = ',filnam
  print *, 'DEBUG: numpts = ',numpts
  print *, 'DEBUG: flotx = ',flotx
  print *, 'DEBUG: fhitx = ',fhitx
  print *, 'DEBUG: florx = ',florx
  print *, 'DEBUG: fhirx = ',fhirx
  print *, 'DEBUG: gauge = ',gauge
  print *, 'DEBUG: length = ',length
*
* Either Read or Generate (then write) the
* discrete transfer function
  if(rdwrit.eq.'WRITE') then
    if(numpts.gt.maxh)then
      write(*,*)'%% Error in CHANL: NUMPTS > ',maxh
      write(i$out,*)'%% Error in CHANL: NUMPTS > ',maxh
      stop
    end if

    delfrq = 1/(dt*numpts)
    nhalf = (numpts/2)+1
    op = .true.
    Funit=20

```

```

do while(op)
  Funit=Funit+1
  inquire(unit=Funit,opened=op)
end do

op = .true.
Munit=21
do while(op)
  Munit=Munit+1
  inquire(unit=Munit,opened=op)
end do
open(unit=Funit,name='Frequency.tmp',status='unknown')
open(unit=Munit,name='Magnitude.tmp',status='unknown')

*
* Compute bandwith and geometric mean frequency
* for the bandpass filter function for both TX and RX transformers.
  W0tx = 2. * pi * sqrt(fhitz*flotx)
  Btx = 2. * pi * (fhitz-flotx)
  W0rx = 2. * pi * sqrt(fhirx*florx)
  Brx = 2. * pi * (fhirx-florx)
*
* Compute the frequency domain transfer function values.

do i=1,nhalf
  freq = (i-1)*delfrq
  omega = 2.0 * pi * freq
  h = BandPass(Btx,W0tx,omega)
  h = TXline(omega,gauge,length)
  h = h * BandPass(Brx,W0rx,omega)
  realh(i) = real(h)
  imgh(i) = aimag(h)
  hmag = cabs(h)
  if (hmag.lt.1.0e-6) then
    hdb = -120.0
  else
    hdb = 20.0*log10(hmag)
  end if

  if ((i.gt.1).and.(i.le.668)) then
    write(Funit,*) freq
    write(Munit,*) hdb
  end if
end do

* we have the transfer funtion from 0 to fs/2, now create from
* fs/2 to fs.(real part has even sym, img part has odd sym).
do i=1,nhalf-2
  realh(nhalf+i) = realh(nhalf-i)
  imgh(nhalf+i) = -imgh(nhalf-i)
end do

```

```

* take inverse FFT of transfer function and scale as necessary
  call fft(realh,imgh,numpts,numpts,numpts,-1)
  do i=1,numpts
    realh(i) = realh(i)/numpts
  end do
*
* resulting impulse response is real, therefore only look at realh.
* Shift impulse response so that the max value is in the center
* of the window (nhalf*dt). This is to prevent the impulse
* response from being split into 2 non-adjacent pieces.
*
  call getmax(realh,numpts,mxindx)
* now that we have the index of the max value of the fir, determine
* the number of units to shift all fir samples.
  nshift = nhalf-mxindx
* shift
  call arshft(realh,numpts,nshift,impres)
* write table out to a formatted data file for NEWTAP
* look for an unused logical unit number
*
  op = .TRUE.
  lun = 20
  do while ( op )
    lun = lun+1
    inquire(unit=lun,opened=op)
  enddo
*
* we have an unused logical unit number. Open a formatted file
* and attach the filename to the logical unit number
*
  open(unit=lun,name=filnam,status='new',
&      form='formatted',err=750)
  goto 760
*
* output error message if OPEN aborts on error
*
750   write(*,*)'%% Error in CHANL when opening ',
      .      FILNAM,' for output.'
      stop
*
760   continue
*
* write out data to NEWTAP since LUN opened successfully
*
  write(lun,*)numpts
  do i=1,numpts
    write(lun,*)impres(i),' ',0.0
  end do

  rewind lun

```



```

        else
*
* if rdwrit=read, then attach a lun to the file,
* then call NEWTAP
      if(rdwrit.ne.'READ') then
        write(*,*) '%% rdwrit defaulted to READ in CHANL'
      end if
*
* look for an unused logical unit number
*
      op = .TRUE.
      lun = 20
      do while ( op )
        lun = lun+1
        inquire(unit=lun,opened=op)
      enddo
*
* we have an unused logical unit number. Attach to file
*
      open(unit=lun,name=filnam,status='old',form='formatted'
&          ,readonly,err=850)
      goto 860
*
* write error message when OPEN aborts on an error.
*
850   write(*,*) '%% Error in CHANL when opening ',
&       FILNAM, ' for input.'
      stop
*
* Come here when OPEN is successful
860   continue
*
* End if READ or WRITE
      end if
*
* End if INIT = YES or NO
      end if
*
* NEWTAP accepts an argument for the logical unit for runtime input
* so we do NOT have to get this data into the standard runtime file
* (SYSTIDATA.DTA). The file that is read is the one generated above;
* ie the impulse response.
*
simulate: CHANNEL *****

      win = cmplx(TX,0.0)
      win < NEWTAP(lun) > wout

```

```

        rcv = real(wout)
*
* close the file with the runtime data for NEWTAP
*
        if ( init .eq. YES ) then
            close(unit=lun)
            init = NO
        endif

END: channel model

```

A.4 D FLIP FLOP

```

MODEL: clr,pr,ck,d > d flipflop(high,low,ref) > q,qbar
|
| This is a D flip flop model similar to the 7474 TTL unit
|
| Written by Tim Davis
|
| Input nodes-
|     clr - reset q output to low.
|     pr  - set q output to high.
|     ck  - clock
|     d   - data
REAL: clr,pr,ck,d
|
| output nodes:
|     q - Data output
|     qbar - complementary output of q
REAL: q,qbar
|
| Parameters
|
|     high - high level output signal value
|     low  - low level output signal value
|     ref  - reference level for logic value determination
REAL high,low,ref
|
| Internal real variables.
|     holdq - Holds latest value of Q output
|     lastck - Holds last value of clock. used for
|              detecting rising edge of clock.
REAL holdq,lastck
STACK <R> holdq,lastck
|

```

```

| Begin simulation
SIMULATE:
  if ((pr.le.ref).and.(clr.gt.ref)) then
    holdq = high

  else if ((pr.gt.ref).and.(clr.le.ref)) then
    holdq = low

  else if ((pr.le.ref).and.(clr.le.ref)) then
    write(i$out,*) '%% FLIP FLOP: clear=preset=low is invalid.'

  else
    |
    | On a rising edge of the clock load D -> Q
    | if ((lastck.le.ref      qbar = low
    | end if
    |
    | lastck = ck

END: of D FLIP FLOP

```

A.5 4-1 MULTIPLEXER

```

MODEL: b,a,s0,s1,s2,s3 > mplexr(high,low,ref) > y,ybar
|
| This model is a 4 to 1 digital multiplexor.
| The two input signals b and a select one of the
| four inputs s0 thru s3 to be regenerated and placed
| on the output line y. (ybar is the complementary output)
| Written by Tim Davis
|
| This model modified Dec 20, 1985
| This model tested xxx xx, xxxx
|
| Input nodes
|   b - MSB of signal selector
|   a - LSB of signal selector
|   s0 thru s1 - input signals to be multiplexed
|
REAL: b,a,s0,s1,s2,s3
|
| Output nodes
|   y - regenerated signal selected from
|       s0 thru s1 by binary number 'ba'
|   ybar - complementary output of y.
|
REAL: y,ybar
|
| Parameters
|   high - high level output value
|   low - low level output value
|

```

```

      |         ref - reference level used for logic value determination
      |
REAL high,low,ref

      |
      | Local vars
REAL s(0:3)
INTEGER i,j,sel

      |
      | Begin
SIMULATE:
      |   i = 0
      |   if (a.gt.ref) i = 1
      |   j = 0
      |   if (b.gt.ref) j = 1
      |   sel = j*2+i
      |   s(0) = s0
      |   s(1) = s1
      |   s(2) = s2
      |   s(3) = s3
      |   y = s(sel)
      |   if (y.gt.ref) then
      |     y = high
      |     ybar = low
      |   else
      |     y = low
      |     ybar = high
      |   end if

END: mplexr

```

A.6 SCHMITT TRIGGER

```

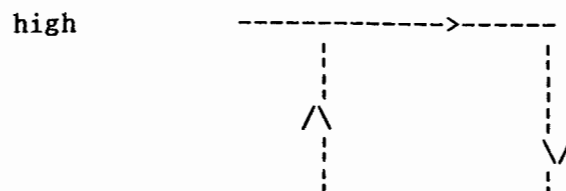
MODEL: sigin > schmitt trigger (high,low,ref,ltl,htl) > sigout

```

This model emulates a 7414 schmitt trigger inverter. IE it has hysteresis on the inputs to reject noise.

This model modified on Dec 19, 1985
This model verified on xxx xx, xxxx

The transfer characteristic appears as follows.



A.7 TIMER

```
MODEL: sign > timer(high,low,ref,timect) > q,qbar
```

```

This model simulates the operation of a one shot.
When the input signal sign changes from less than
the reference level to greater than the reference level,
q is set high for timect*dt seconds.
When the timer is "on", the value of sign is ignored
so retriggering is not possible.
Outbar is the complementary output of q.

```

```
This model modified Dec 19, 1985
```

```
This model tested xxx xx, xxxx
```

```
If timect < 1 then the program is aborted
```

```
REAL: sign, q,qbar
```

```
Parameters
```

```
high - output level when "signal" > ref
```

```
low - output level when "signal" < ref
```

```
ref - reference level used for logic determination
```

```
timect - Integral number of DT's to keep q at high level
after rising edge of sign.
```

```
REAL high,low,ref
```

```
INTEGER timect
```

```
Internal variables
```

```
cntr - Counts DT's while timer is on.
```

```
When 0 it means timer is off.
```

```
lstin - Value of sign from last call to model
```

```
REAL lstin
```

```
STACK <real> lstin
```

```
INTEGER cntr
```

```
STACK <integer> cntr
```

```
Begin
```

```
CALCULATE:
```

```
If (timect.lt.1) then
```

```
print *,'!!! Timer aborted. Timect < 1'
```

```
stop
```

```
end if
```

```

SIMULATE: timer
  if (cntr.gt.0) then
    q = high
    qbar = low
    cntr = cntr + 1
    if (cntr.eq.timect) cntr = 0

; timer is off. Check for rising edge
else
  if ((lstin.le.ref).and.(sigin.gt.ref)) then
    cntr = 1
    q = high
    qbar = low

; No rising edge, keep timer off.
else
  q = low
  qbar = high
end if

end if

lstin = sigin

END: timer

```

A.8 RANDOM PULSE GENERATOR

```

MODEL: clock > ranpls(high,low,ref,iseed) > rout

```

```

| This model generates a random pulse sequence using
| the uniform distribution random number generator
| RAN(i) in VAX-11 fortran. A new pulse is started on the rising
| edge of clock and terminates one DT before the following
| rising edge of clock. This guarantees that the clock and
| data coincide.

```

```

| This model modified on Dec 27, 1985
| This model verified on Dec 27, 1985

```

```

| Input Signal

```

```

|         clock - Signal which follows the logic conventions
|                 dictated by ref and high, low

```

```

REAL: clock

```

```

| Output signal

```

```

|         rout - Random pulse sequence with pulse widths
|                 equal to period of clock signal

```

REAL: rout

Parameters

high - Logic high output value
 low - Logic low output value
 ref - Reference level for determining logic level.
 sig.gt.ref is high, sig.le.ref is low
 iseed - Random number seed for the RAN() function.
 *** WARNING ***
 Do not call RANPLS with a constant.
 Always use an integer variable;
 It is intended that a value be returned.

REAL high,low,ref
 INTEGER iseed

Internal variables

lstclk - The value of clock from the last call to RANPLS()
 hold - The output value for ROUT. Maintained for entire
 bit time of clock until a new rising edge occurs.

REAL lstclk,hold
 STACK <R> lstclk,hold

SIMULATE: random pulse source
 if ((lstclk.le.ref).and.(clock.gt.ref)) then
 if (ran(iseed).ge.0.5) then
 hold = high
 else
 hold = low
 end if
 end if

 rout = hold
 lstclk = clock

 END: random pulse source

A.9 ABSOLUTE VALUE

MODEL: in > abs value > out
 REAL: in,out

SIMULATE:
 out = abs(in)
 END:

A.11 TRIMPOWER.FOR

```

      program TrimPower
c
c This program trims away a user specified amount
c of power from the tails of impulse reponses generated
c by the SYSTID simulation DAVIS07.TXT
c
      real          max,data,totalpower,tailpower,epsilon
      dimension     data(4096)
      integer       samples,i,position,front,rear
      character*50  file,trimfile
      logical       trim

      totalpower = 0.
      max = -1.0e32
      print *, '> Enter impulse response file name'
      accept 1,file
1      format(a50)
      open(unit=1,name=file,access='sequential',
&         status='Unknown')
      read(1,*) samples
      do i=1,samples
          read(1,*) data(i)
          totalpower = totalpower + data(i)*data(i)
          if (data(i).gt.max) then
              max = data(i)
              position = i
          end if

          end do

      close(unit=1)
      print *, '* Maximum value occurs at ',position
      print *, ' Maximum value is ',max

      trim = .true.
      do while (trim)
          print *, '> Enter percentage tail power to remove'
          accept *,epsilon
          front = 0
          rear = samples + 1
          tailpower = 0.
          do while (tailpower/totalpower*100.0.le.epsilon)
              rear = rear - 1
              front = front + 1
              tailpower = tailpower +
&                          data(front)*data(front)+
&                          data(rear)*data(rear)
          end do
      end do

```

```

c
c err on the side of removing to little power.
      tailpower = tailpower - data(front)*data(front)-
&          data(rear)*data(rear)
      rear = rear + 1
      front = front -1

      print *,'* The impulse response is being trimmed'
      print *,' to samples ',front,'to',rear,
&          ': count=',rear-front+1
      print *,'* Total power left is ',
&          100.-tailpower/totalpower*100.

      print *, '> Do you want to try another ',
&          'tail power trim percentage?'
      print *,' enter .true. or .false.'
      accept *,trim
      end do

      print *, '> Enter name of file to store trimmed ',
&          'impulse response to'
      accept 1,trimfile
      open(unit=2,name=trimfile,access='sequential',
&          status='unknown')
      write(2,2) rear-front+1,100.-epsilon,file
2      format(1x,i11,10x,'This file contains ',f4.1,
&          '% power from file ',a50)
      do i=front,rear
          write(2,3) data(i),0.
      end do
3      format(1x,e14.7,4x,e14.7)
      close(unit=2)

      stop
      end

```

A.12 BANDPASS.FOR

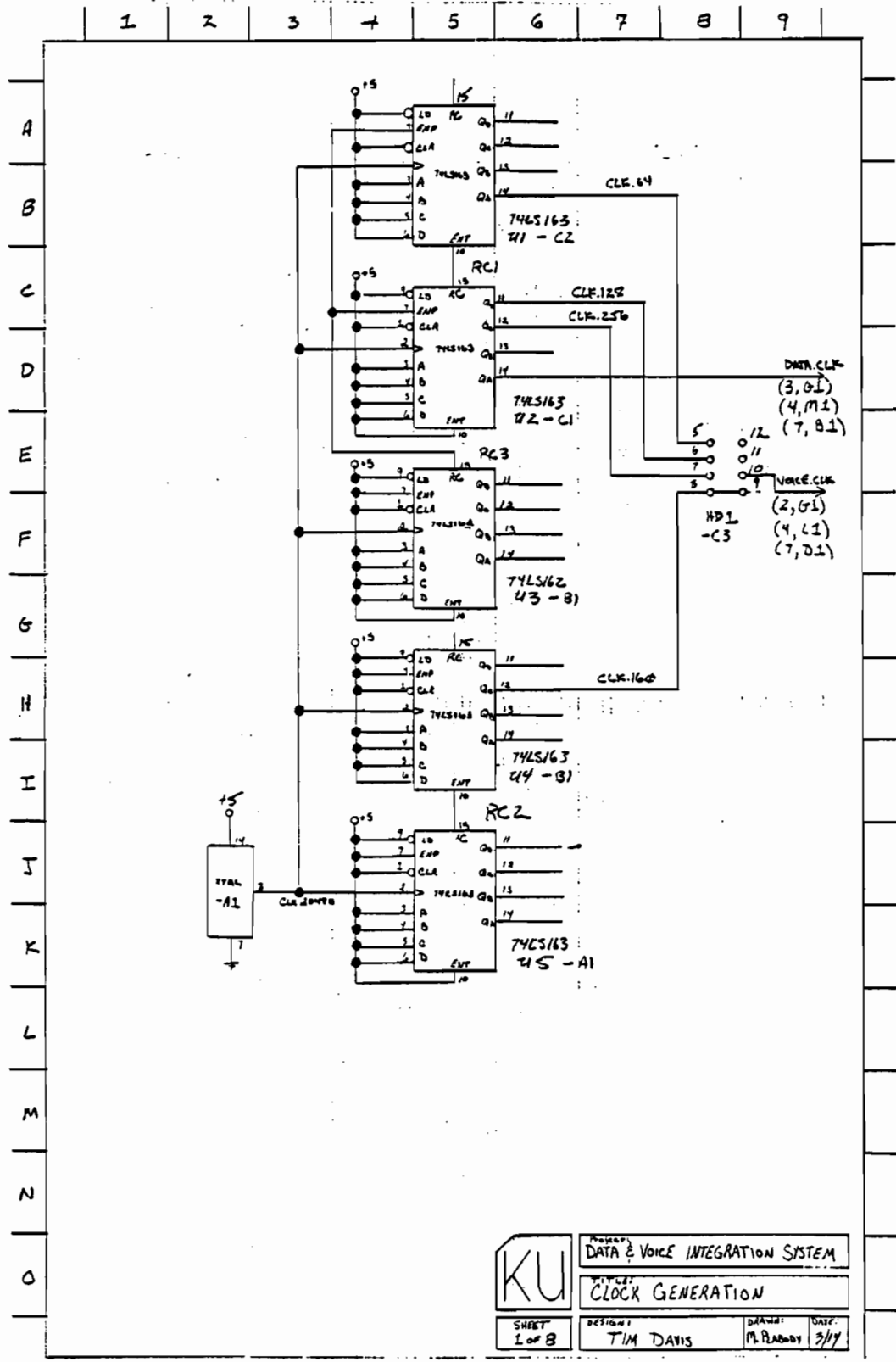
```

      complex function BandPass(B,W0,omega)
c
c The transformers are modeled as second order butterworth
c bandpass filters. The transfer function for each transformer
c is obtained by transforming the normalized second order
c butterworth lowpass filter transfer function
c
c       $H_n(s) = 1/(s^2 + 1.4142*s + 1)$ 
c
c by substituting 's' with the 'p' given below.
c
c       $p = W0/B * ( s/W0 + W0/s)$ 
c
      complex      p
      real      omega,B,W0
      if (omega.lt.1.0e-6) then
          BandPass = (0.0,0.0)
      else
          p = W0/B * (cplx(0.,omega)/W0+W0/cplx(0.,omega))
          BandPass = 1.0/(p*p+1.4142*p+1)
          end if

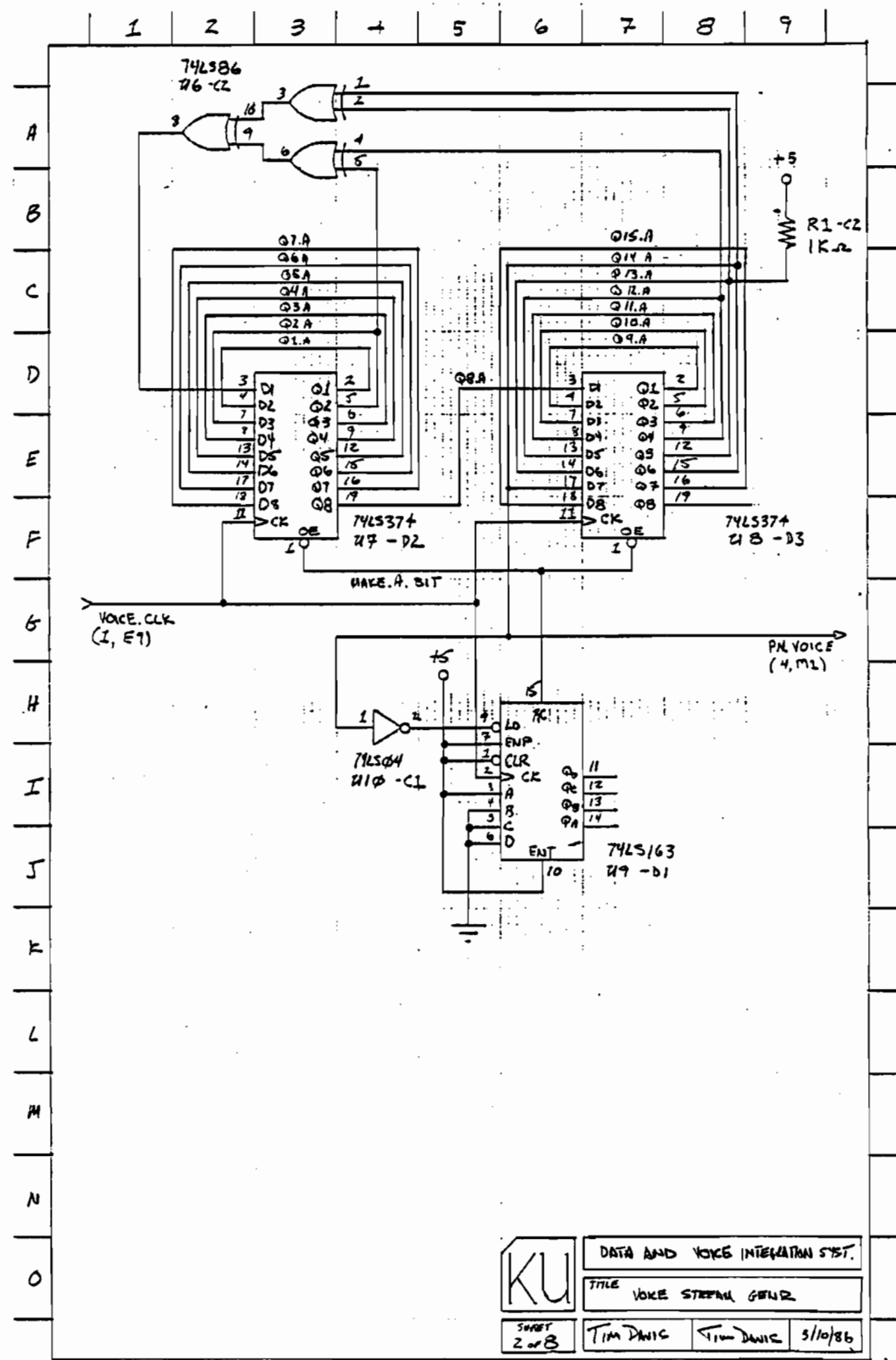
      return
      end

```

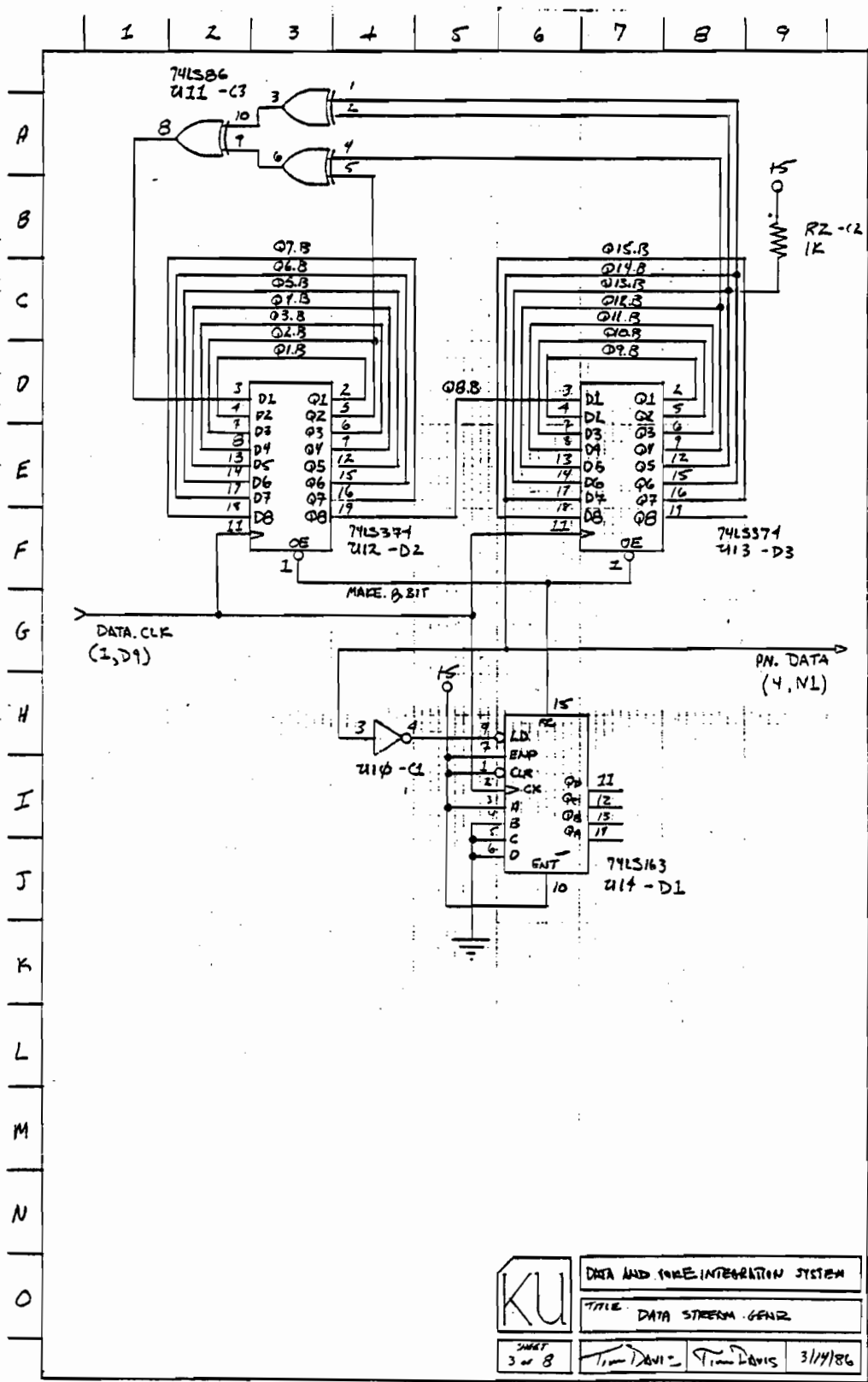
B. PROTOTYPE CIRCUIT SCHEMATICS



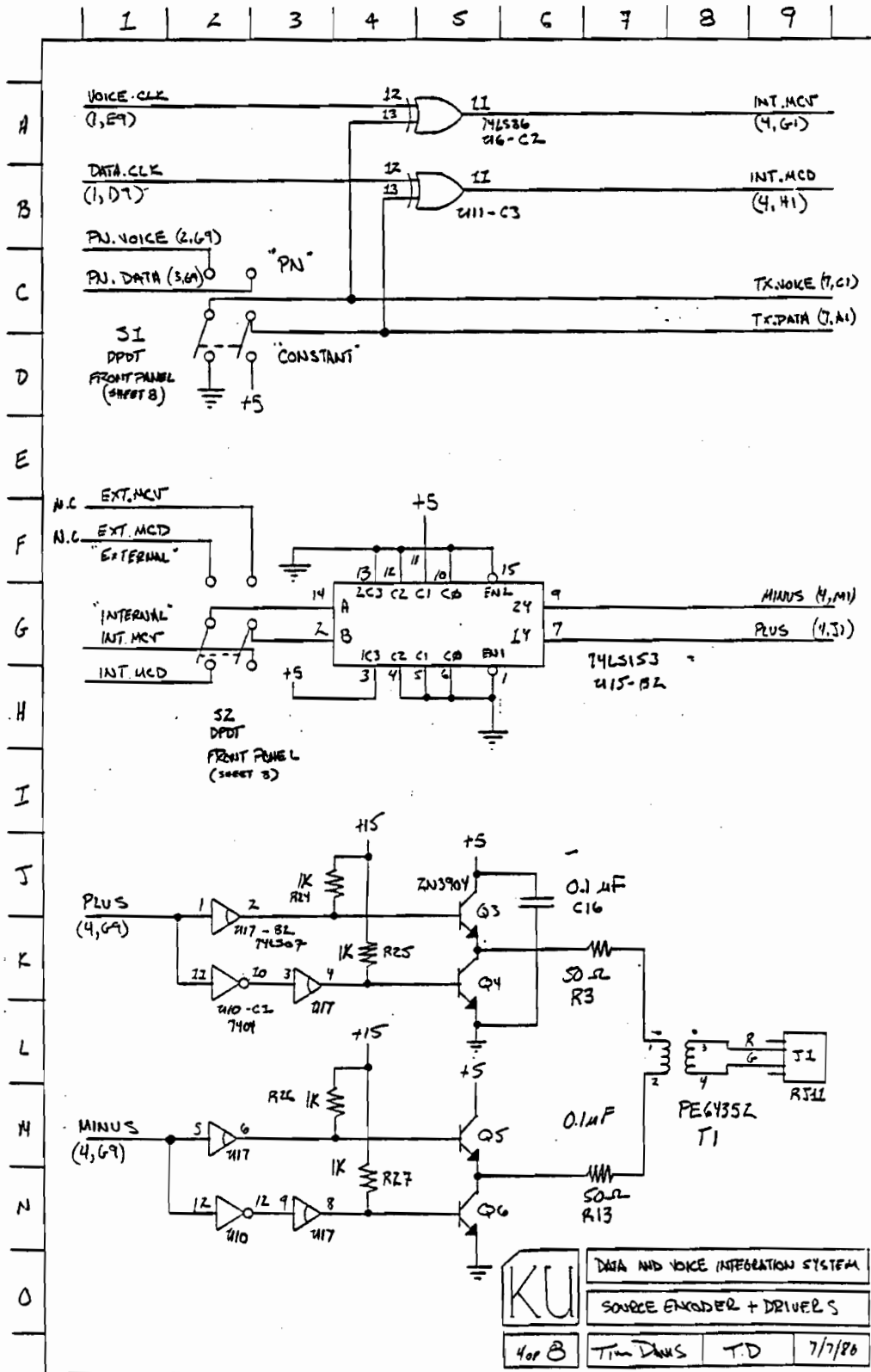
	PROJECT: DATA & VOICE INTEGRATION SYSTEM	
	TITLE: CLOCK GENERATION	
SHEET: 1 of 8	DESIGNER: TIM DAVIS	DRAWN: M. RABOBY DATE: 3/17

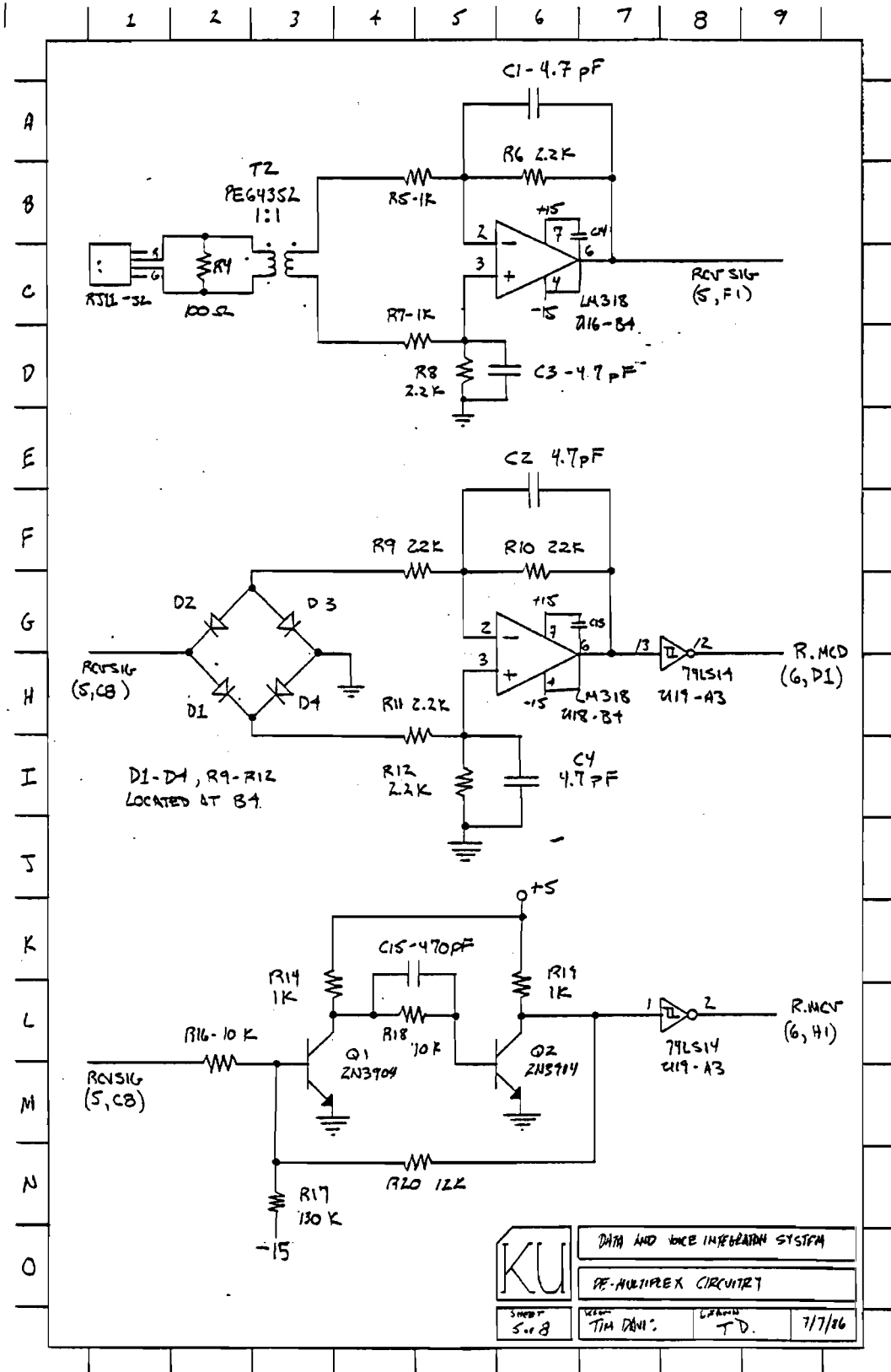


KU	DATA AND VOICE INTERACTION SYST.		
	TITLE VOICE STREAM GENER.		
Sheet 2 of 8	TIM DANIC	TIM DANIC	5/10/86

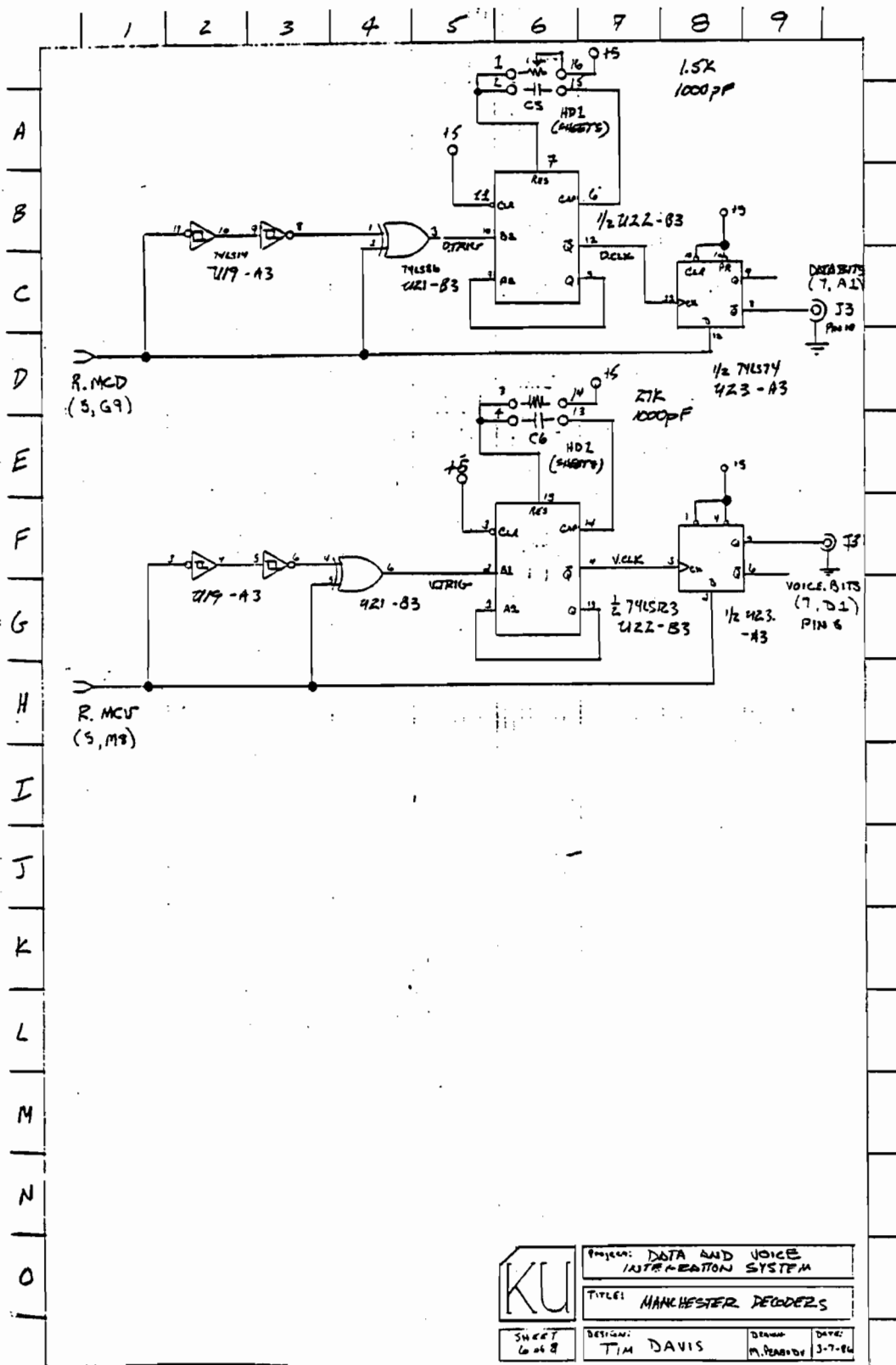


	DATA AND TONE INTEGRATION SYSTEM
	TITLE: DATA STREAM GENER
	SHEET 3 of 8 Tim Davis - Tim Davis 3/14/86

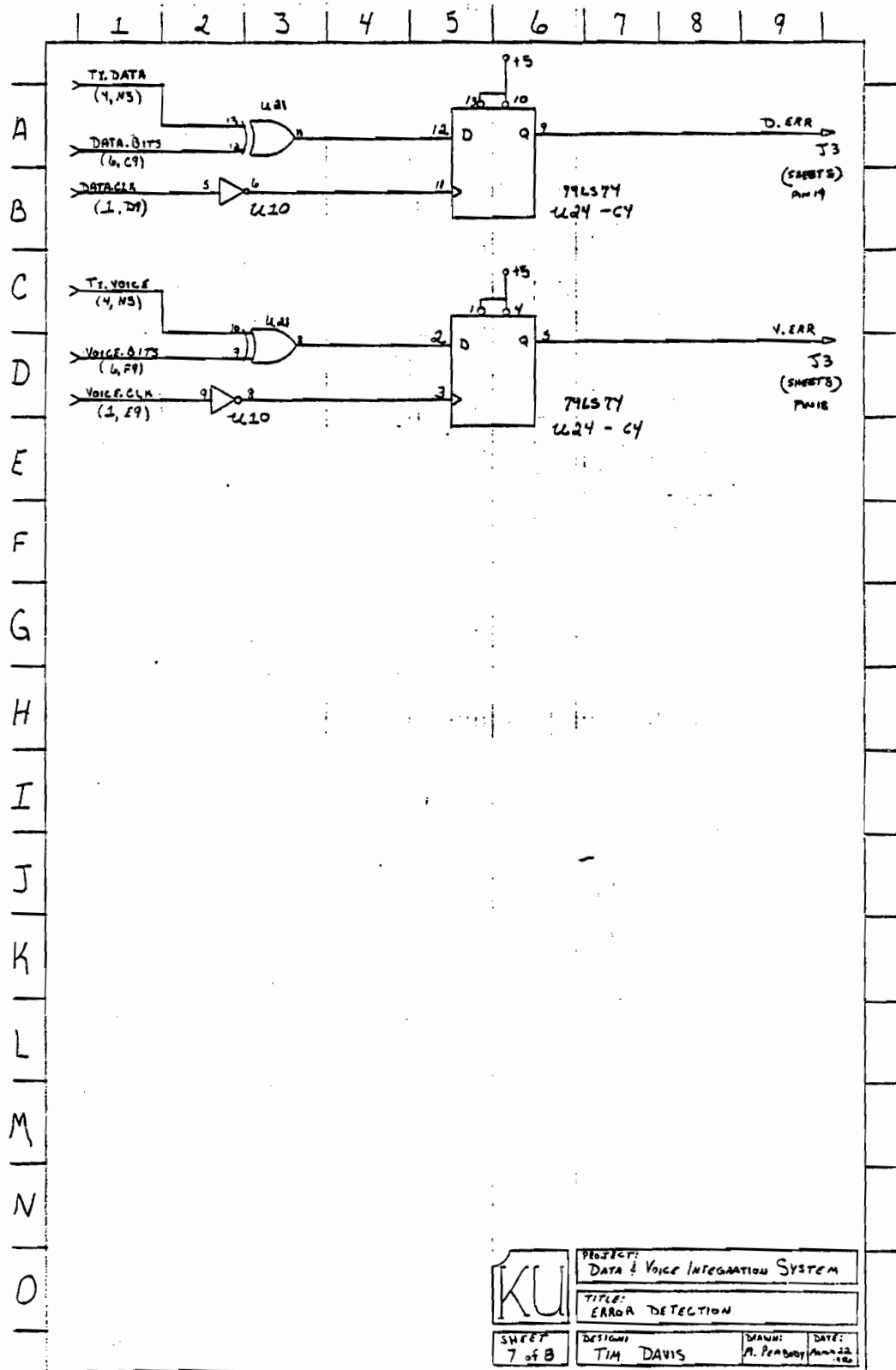




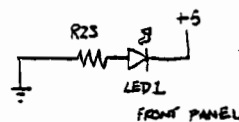
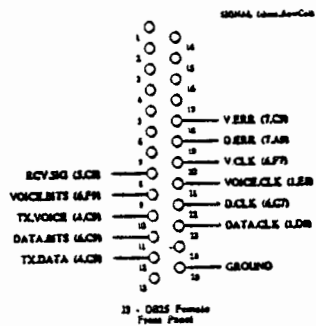
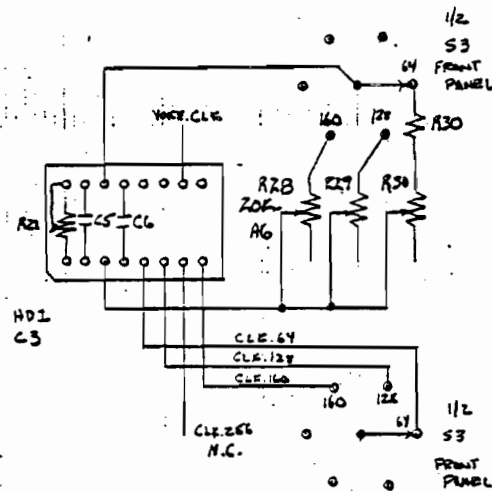
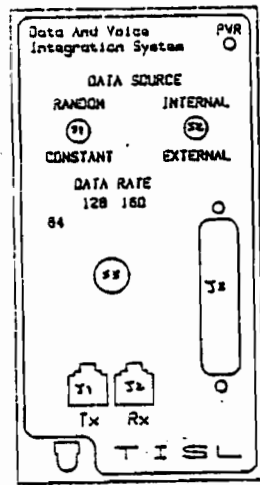
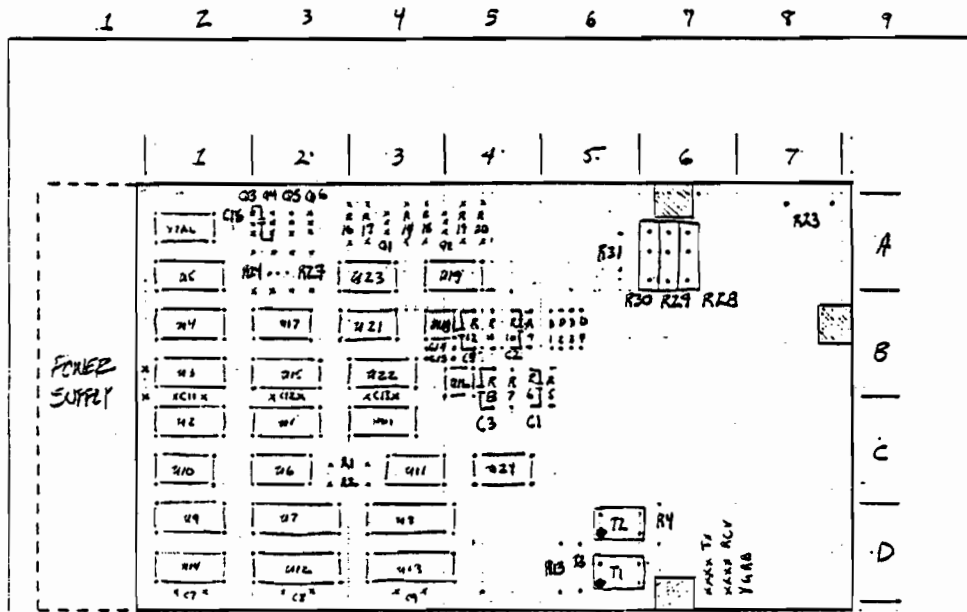
	DATA AND VOICE INTERFERENCE SYSTEM		
	PE-MULTIPLEX CIRCUITRY		
SHEET 5 of 8	DESIGNED TIM DANIELS	CAPTURED T.D.	DATE 7/7/86



	Project: DATA AND VOICE INTER-RELATION SYSTEM		
	TITLE: MANCHESTER DECODERS		
SHEET 6 of 8	DESIGNER: TIM DAVIS	DRAWN: M. REARDON	DATE: 3-7-64



KU	PROJECT: DATA & VOICE INTEGRATION SYSTEM		
	TITLE: ERROR DETECTION		
SHEET 7 of 8	DESIGNER: TIM DAVIS	DRAWN: M. Penbury	DATE: March 22, 1980



KU	Project DATA & VOICE INTEGRATION SYSTEM		
	TITLE: TMS02 BOARD LAYOUT		
SHEET 8 of 8	DESIGNER: Tim DENNIS	DRAWN: T.D.	DATE: 3/11

C. PARTS LOCATER

I N T E G R A T E D C I R C U I T S

Component ID	Part Number	Board RowCol	location	
			Schematic (Sheet,RowCol)	
U1	741s163	C2	(1,B5)	
U2	741s163	C1	(1,D5)	
U3	741s162	B1	(1,F5)	
U4	741s163	B1	(1,H5)	
U5	741s163	A1	(1,J5)	
U6	741s86	C2	(2,A2)	
			(4,A5)	
U7	741s374	D2	(2,D3)	
U8	741s374	D3	(2,D7)	
U9	741s163	D1	(2,H5)	
U10	741s04	C1	(2,H4)	
			(3,H4)	
			(4,K2)	
			(4,N2)	
			(7,B2)	
			(7,D2)	
U11	741s86	C3	(3,A2)	
			(4,B5)	
U12	741s374	D2	(3,D3)	
U13	741s374	D3	(3,D7)	
U14	741s163	D1	(3,H6)	
U15	741s153	B2	(4,G4)	
U16	LM318	B3	(5,C6)	
U17	741s07	B2	(4,J2)	
			(4,K3)	
			(4,M2)	
			(4,N3)	
U18	LM318	B4	(5,H6)	
U19	741s14	A3	(5,L8)	
			(5,G8)	
			(6,B2)	
			(6,B3)	
			(6,F2)	
			(6,F3)	
U20	Discarded			
U21	741s86	B3	(6,C4)	
			(6,G4)	
			(7,A3)	
			(7,D3)	
U22	741s123	B3	(6,G6)	

I N T E G R A T E D C I R C U I T S

Component ID	Part Number	Board RowCol	location
			Schematic (Sheet,RowCol)
U23	741s74	A3	(6,C8)
			(6,G8)
U24	741s74	C4	(7,A5)
			(7,D5)

M I S C E L L A N E O U S C O M P O N E N T S

Component ID	Part Number	location	
		Board RowCol	Schematic (Sheet,RowCol)
XTAL	CC0100A:20.48	A1	(1,J2)
HD1	header	C3	(1,E8)
			(6,A6)
			(6,E6)
			(8,I4)
S1	DPDT	Front Panel	(4,C2)
S2	DPDT	Front Panel	(4,G2)
S3	DP 6 throw	Front Panel	(8,H7)
			(8,K7)
LED1	Red LED	Front Panel	(8,M6)
T1	PE64352	D5	(4,L8)
T2	PE64352	D5	(5,C3)
J1	RJ11	Front Panel	(4,L9)
J2	RJ11	Front Panel	(5,C1)
J3	DB25	Front Panel	(8,N3)

RESISTORS

Component ID	Part Number	Board RowCol	location
			Schematic (Sheet,RowCol)
R1	1k	C2	(1,B9)
R2	1k	C2	(2,B9)
R3	50	D5	(4,K7)
R4	100	D6	(5,C2)
R5	1k	B5	(5,B4)
R6	2.2k	B4	(5,B6)
R7	1k	B4	(5,D4)
R8	2.2k	B4	(5,D4)
R9	2.2k	B4	(5,F4)
R10	2.2k	B4	(5,F6)
R11	2.2k	B4	(5,H4)
R12	2.2k	B4	(5,I4)
R13	50	D5	(4,N6)
R14	1k	A3	(5,L3)
R15	Unused		
R16	10k	A3	(5,L2)
R17	130k	A3	(5,N3)
R18	10k	A4	(5,L4)
R19	1k	A4	(5,L6)
R20	12k	A5	(5,N4)
R21	unused		
R22	5k pot	C3	(8,I4) (6,A6)
R23	470	A7	(8,M4)
R24	1k	A2	(4,J4)
R25	1k	A2	(4,K4)
R26	1k	A2	(4,M2)
R27	1k	A2	(4,N4)
R28	20k pot	A6	(8,I6)
R29	20k pot	A6	(8,I7)
R30	20k pot	A6	(8,I7)
R31	5.6k	A5	(8,H8)

C A P A C I T O R S

Component ID	Part Number	location	
		Board RowCol	Schematic (Sheet,RowCol)
C1	4.7pf	B5	(5,A6)
C2	4.7pf	B4	(5,E6)
C3	4.7pf	B4	(5,D6)
C4	4.7pf	B4	(5,I6)
C5	1000pf	C3	(6,A6)
			(8,I4)
C6	1000pf	C3	(6,E6)
			(8,I5)
C7	0.1uf	D1	
C8	0.1uf	D2	
C9	0.1uf	Bypass Cap	
C10	0.1uf		
C11	0.1uf		
C12	0.1uf		
C13	0.1uf		
C14	0.1uf	B3	(5,C7)
C15	0.1uf	B3	(5,G7)
C16	0.1uf	A2	(4,J6)

S E M I C O N D U C T O R S

Component ID	Part Number	location	
		Board RowCol	Schematic (Sheet,RowCol)
D1		B5	(5,H2)
D2		B5	(5,G2)
D3		B5	(5,G3)
D4		B5	(5,H3)
Q1	2N3904	A3	(5,L3)
Q2	2N3904	A3	(5,L6)
Q3	2N3904	A2	(4,J5)
Q4	2N3904	A2	(4,K5)
Q5	2N3904	A2	(4,M5)
Q6	2N3904	A2	(4,N5)

D. SIGNAL LOCATER

SIGNAL GLOSSARY

Signal Name	Origin IC/Pin	Location Board Row Col	Schematic (Sht,Grid)	Description
CLK.64	U1/14	C2	(1,A5)	64 kHz square wave used to clock voice PN sequence generator.
CLK.128	U2/11	C1	(1,C5)	128 kHz square wave used to clock voice PN sequence generator.
CLK.256	U2/12	C1	(1,D5)	256 kHz square wave used to clock voice PN sequence generator.
CLK.160	U4/12	B1	(1,H5)	160 kHz square wave used to clock voice PN sequence generator.
CLK.20480	XTAL/8	A1	(1,J2)	20.48 MHz master system clock.
DATA.CLK	U2/14	C1	(1,D5)	1024 kHz square wave used to clock data PN sequence generator.
VOICE.CLK	HD1/9	C3	(1,E8)	This is one of four clock signals (64,128,160,256) that drives the voice PN sequence generator.
MAKE.A.BIT	U9/15	D1	(2,H6)	Set high when a string of 14 zeroes is about to occur in the voice PN shift register.
MAKE.B.BIT	U14/15	D1	(3,H6)	

Signal Name	Origin IC/Pin	Location Board Row Col	Schematic (Sht,Grid)	Description
				Set high when a string of 14 zeroes is about to occur in the data PN shift register.
PN.VOICE	U8/16	D3	(2,F8)	Pseudo random bit stream clocked by VOICE.CLK.
PN.DATA	U13/16	D3	(3,F8)	Pseudo random bit stream clocked by DATA.CLK.
INT.MCV	U6/11	C2	(4,A5)	Internal source (from PN.VOICE) manchester encoded.
INT.MCD	U11/11	C3	(4,B5)	Internal source (from PN.DATA) manchester encoded.
MINUS	U15/9	B2	(4,G6)	Output from DVM multiplexer which goes to negative input at differential receiver.
PLUS	U15/7	B2	(4,G6)	Output from DVM multiplexer which goes to positive input at differential receiver.
RCVSIG	U16/6	B4	(5,C7)	Recovered three level DVM signal which needs to be split into voice and data.
RCVABS	U18/6	B3	(5,G7)	Absolute value of RCVSIG which yields data stream.
R.MCD	U19/12	A3	(5,G8)	Recovered manchester encoded data stream.
R.MCV	U19/2	A3	(5,L8)	Recovered manchester encoded voice stream.

Signal Name	Origin IC/Pin	Location Board Row Col	Schematic (Sht,Grid)	Description
D.TRIG	U21/3	B3	(6,B5)	Trigger pulses at edge transitions of recovered manchester data (R.MCD).
V.TRIG	U21/6	B3	(6,F5)	Trigger pulses at edge transitions of recovered manchester voice (R.MCV).
D.CLK	U22/12	B3	(6,C7)	Recovered data clock for sampling incoming manchester (R.MCD) to get NRZ data.
V.CLK	U22/4	B3	(6,F7)	Recovered voice clock for sampling incoming manchester (R.MCV) to get NRZ voice.
DATA.BITS	U23/8	A3	(6,C7)	NRZ data bits recovered from demultiplexed manchester data.
VOICE.BITS	U23/5	A3	(6,F7)	NRZ voice bits recovered from demultiplexed manchester voice.
D.ERR	U24/9	C4	(7,A6)	Signals an error found in DATA.BITS.
V.ERR	U24/5	C4	(7,D6)	Signals an error found in VOICE.BITS.